COMP 433 Lecture 2

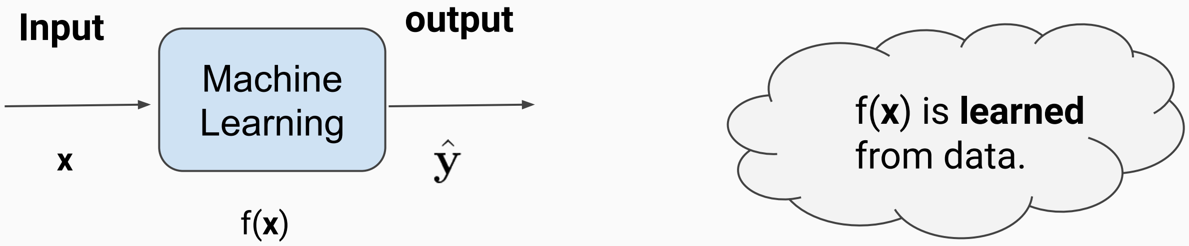
Practical Matters

- Lab 2 will be posted today
  - Lab 1 will not count since most of students are familiar with Numpy and Pytorch
  - Lab 5 will take the grade of Lab 1
- Final Syllabus will be available on Monday (with likely dates of Quizzes and Assignments)

- Today Slide Credit: Mirco Ravanelli

What is machine learning?

Machine learning aims to build machines that **learn from data** (or, more in general, from **experience**).



- We want a function *f(x)* that maps the input *x* into the desired output *y*:

$$\boxed{\hat{\mathbf{y}} = f(\mathbf{x})} \quad \mathbf{x} \in \mathbb{R}^D, \hat{\mathbf{y}} \in \mathbb{R}^K \quad f : \mathbb{R}^D \to \mathbb{R}^K$$

- In practice, we show to machine many **input-output examples**. The algorithm should learn the mapping between the input and output spaces with these examples.

Types of Problems

Classification: the machine learning algorithm has to specify which of *k* categories some input belongs to.

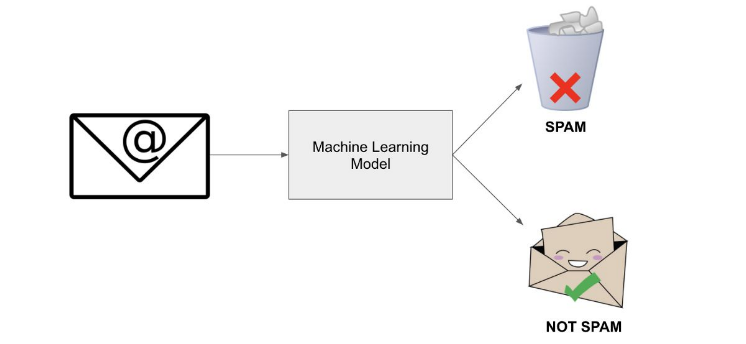
$$\boxed{\hat{y} = f(\mathbf{x})} \qquad \mathbf{x} = [x_1, x_2, .., x_D]^T$$

Row-vector
(by convention)

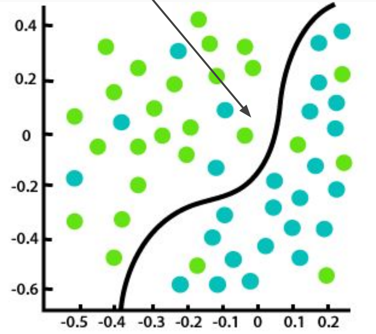
$$\mathbf{x} \in \mathbb{R}^D \qquad \hat{y} = \{1, \ldots, K\}$$

Decision boundaries

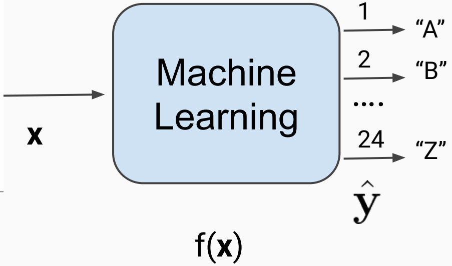
f(**x**)



X2



X1



Types of Problems

Regression: the machine learning algorithm predicts a continuous value given some input.

$$\boxed{\hat{y} = f(\mathbf{x})} \quad \mathbf{x} = [x_1, x_2, .., x_D]^T \quad \mathbf{x} \in \mathbb{R}^D \quad \hat{y} \in \mathbb{R}$$

$$f : \mathbb{R}^D \rightarrow \mathbb{R}$$

Row-vector
(by convention)

f(**x**)

Number of floors: **2**

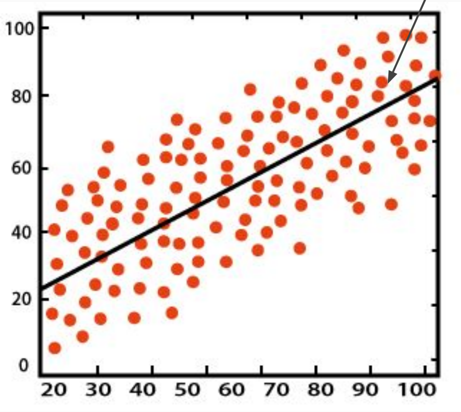
Number of bedrooms: **3**

Age: **15**

$\mathbf{x}$ → Machine Learning → **Price** $\hat{y}$ → *665k$*

f(**x**)

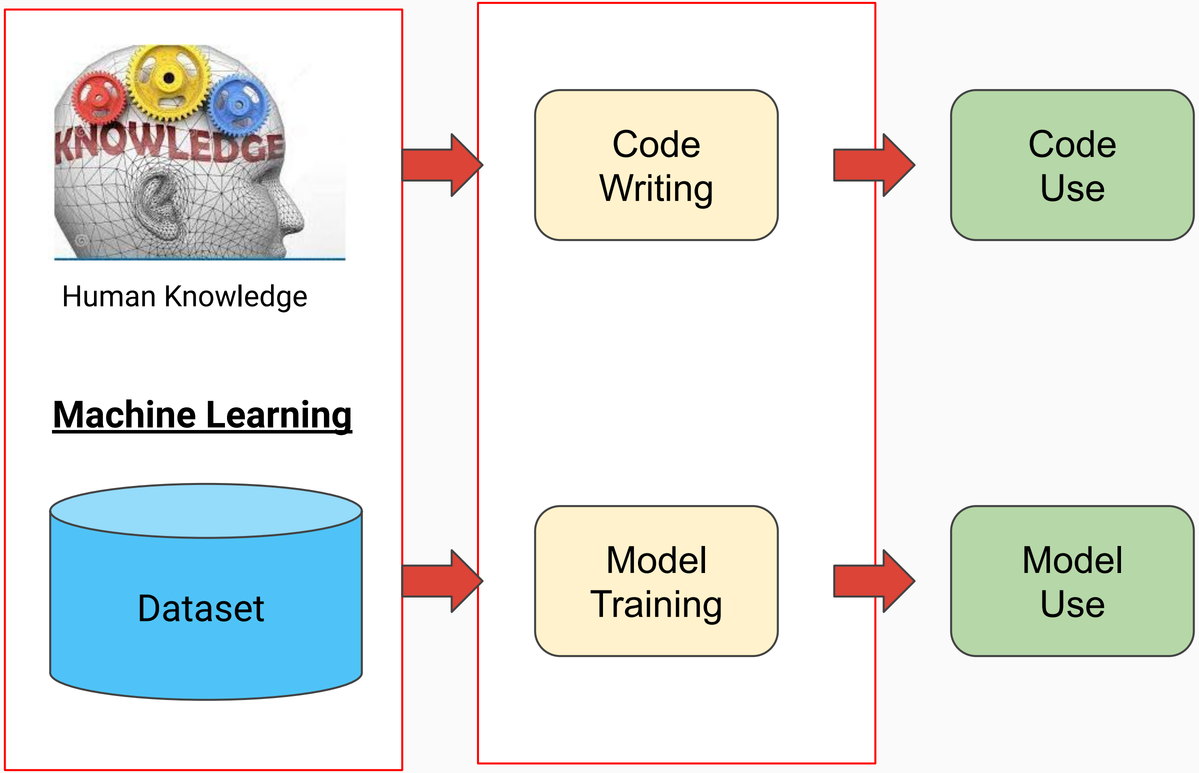
X2



X1

Traditional Programming vs Machine Learning

Traditional Programming



Main Differences:

- With machine learning, we replace **human knowledge** with **data**.
- We replace the hand-crafted code with a function f(**x**) **learned from data**.

Traditional Programming vs Machine Learning

Example: max value of a list

Traditional Programming

Human Knowledge

```python
def get_max(lst):
  max_value = - math.inf
  for elem in lst:
      if elem > max_value:
        max_value = elem
  return max_value
```

Code Writing

```python
lst = [2, 10, 5, 6]
print(get_max(lst))
```

```
10
```

Code Use

Machine Learning

```python
inputs  = [[2, 10, 5, 6],
           [1, 4, -5, 6],
           [-1, 0, 0, 1],
           [2, 11, 2, 2]
           ]
```

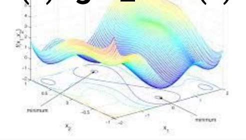
Targets = [10, 6, 1, 11]

Dataset

Model Training

Machine Learning

f(x)=**get_max**(x)



```python
lst = [2, 10, 5, 6]
print(get_max(lst))
```

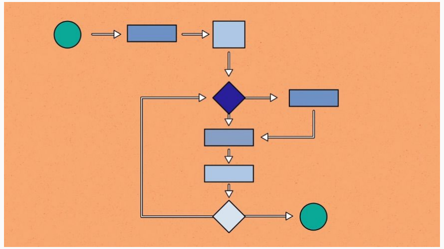
```
10
```

Model Use

Traditional Programming vs Machine Learning

When using traditional programming

- If the problem can be efficiently solved with a well-defined algorithm.



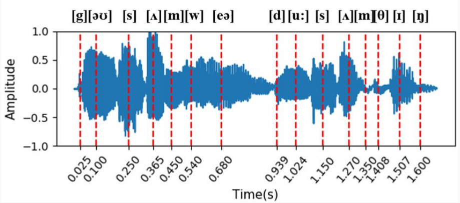
e.g, sorting, search, hashing algorithms.

When using machine learning

- If the problem cannot be solved with a list of formal rules.

- It is easy to collect data to solve the problem.

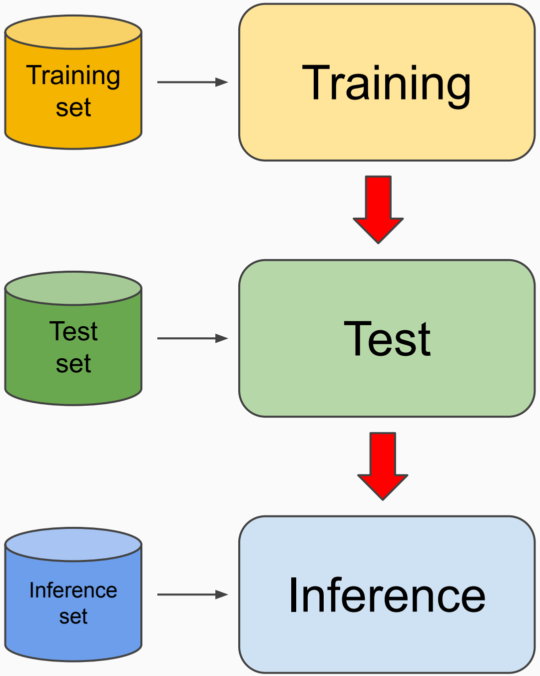
e.g., face recognition, speech recognition



Machine Learning Stages

Machine Learning Stages

- A machine learning algorithm typically goes through the following **stages**:

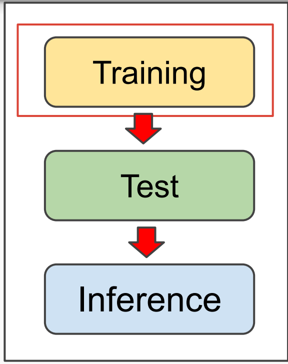


Training: Learn a function that “explains” well the training data.

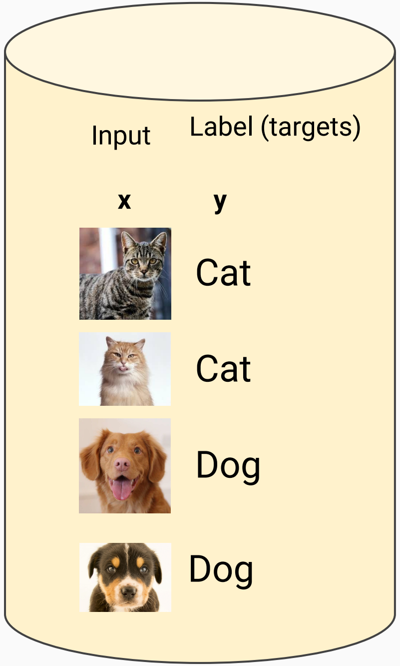
Test: Evaluate the performance on data never seen before (test set).

Inference: Use the model (production).

Classification Example

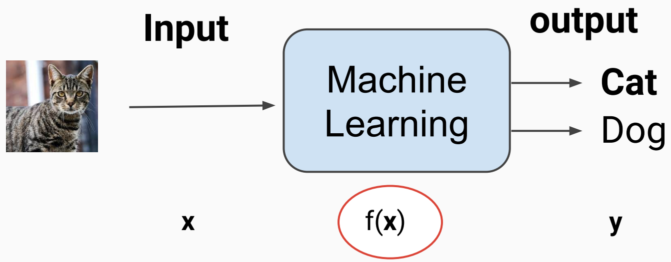


Example: Cat vs Dog classification.



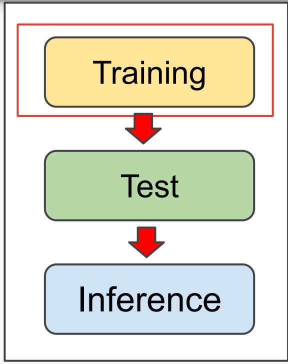
Training Set

- Phase 1: **Training** (Learning)

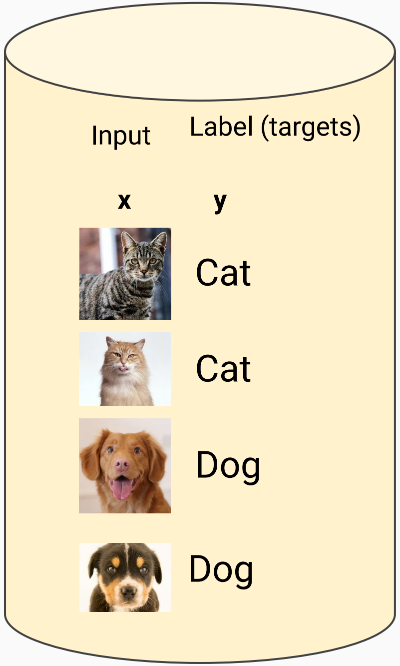


- We show to the machine the input-output examples collected in a **training set**.
- The goal of training is finding f(x) such that it "*explains well*" the training data.

Classification Example

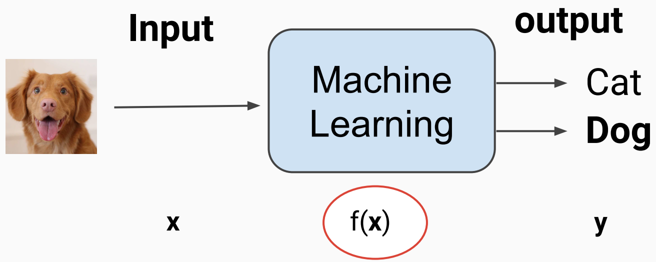


Example: Cat vs Dog classification



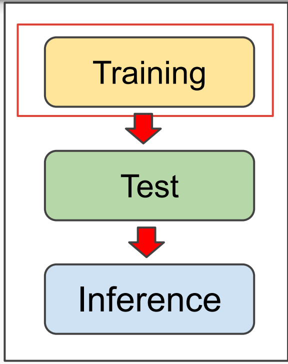
Training Set

- Phase 1: **Training** (Learning)

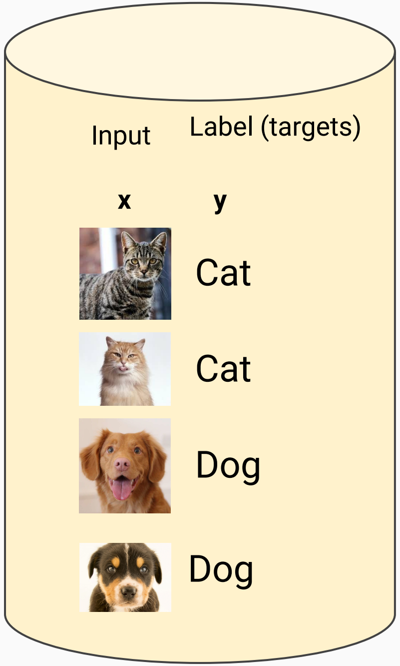


- We show to the machine the input-output examples collected in a **training set**.
- The goal of training is finding f(x) such that it "*explains well*" the training data.

Classification Example

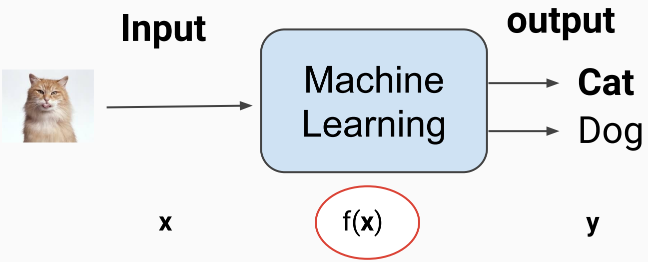


Example: Cat vs Dog classification



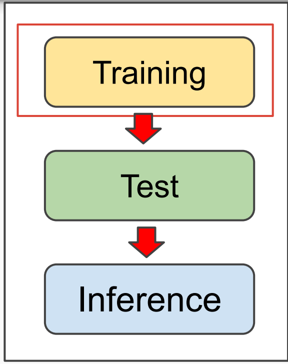
Training Set

- Phase 1: **Training** (Learning)

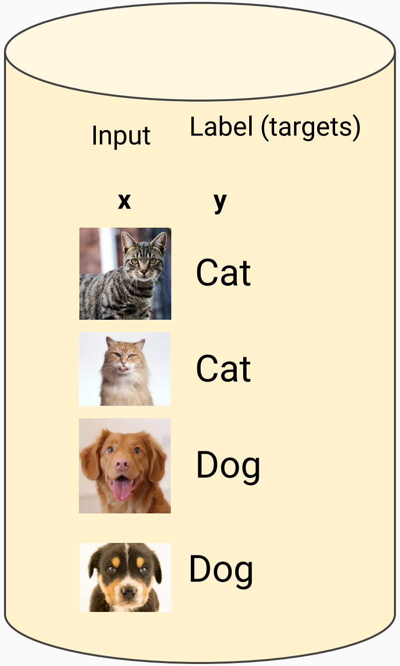


- We show to the machine the input-output examples collected in a **training set**.
- The goal of training is finding f(x) such that it "*explains well*" the training data.

Classification Example

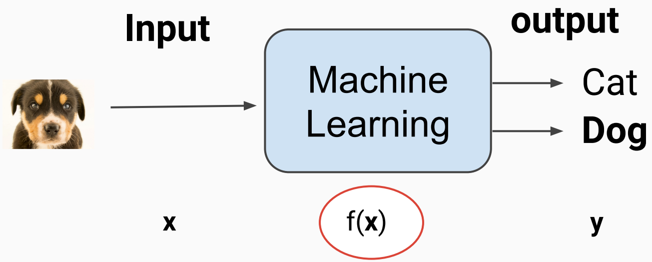


Example: Cat vs Dog classification



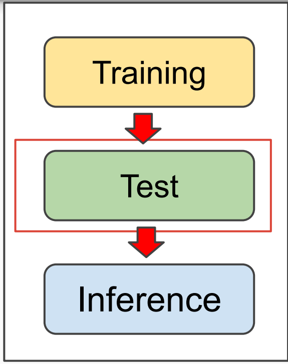
Training Set

- Phase 1: **Training** (Learning)

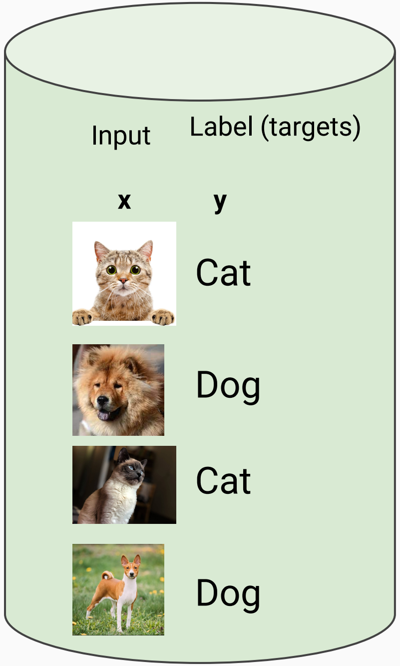


- We show to the machine the input-output examples collected in a **training set**.
- The goal of training is finding a f(x) such that it "*explains well*" the training data.

Classification Example

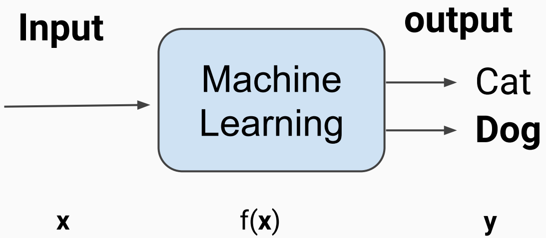


Example1 : Cat vs Dog classification



Test Set

- Phase 2: **Test** (Evaluation)

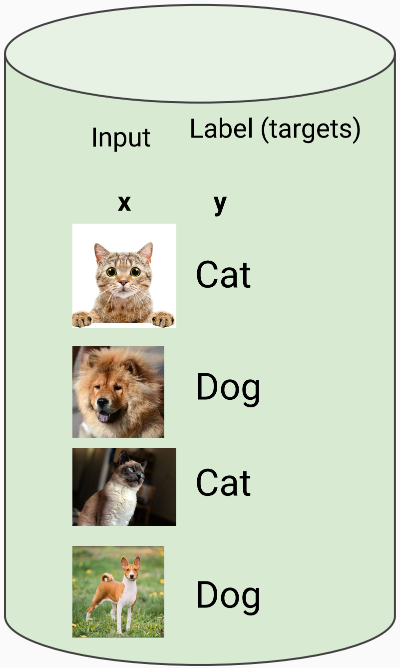


- A good machine learning model should perform well on data never seen before. This ability is called **generalization**.

- The goal of the testing phase (evaluation) is to **measure the performance** on data never seen before (collected in a test set).

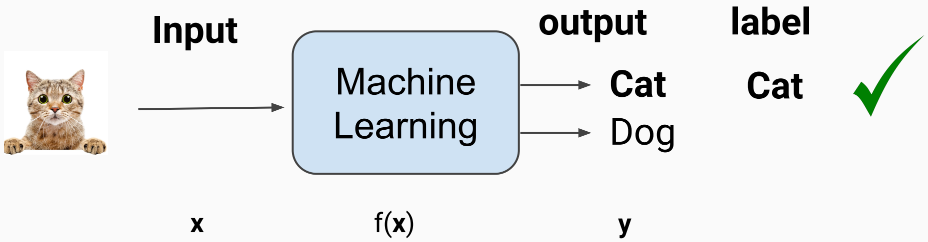
Classification Example

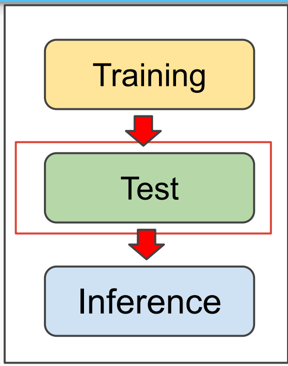
Example: Cat vs Dog classification



Test Set

- Phase 2: **Test** (Evaluation)



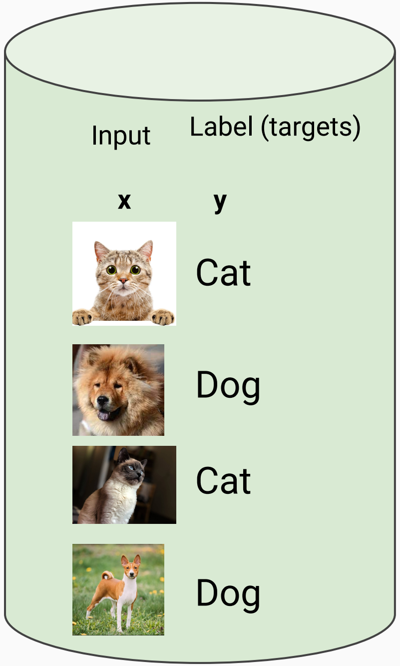


- A good machine learning model should perform well on data never seen before. This ability is called **generalization**.

- The goal of the testing phase (evaluation) is to **measure the performance** on data never seen before (collected in a test set).

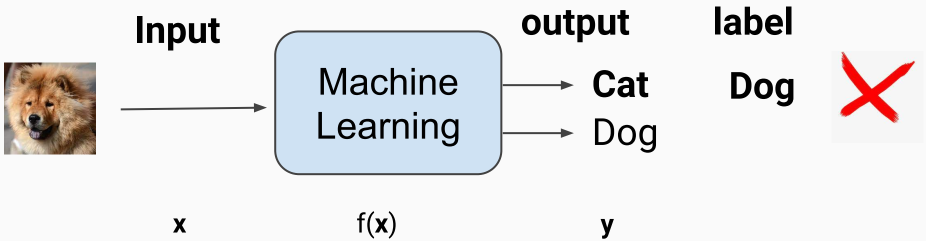
Classification Example

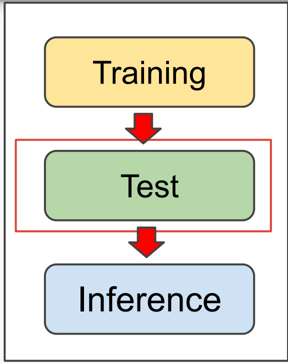
Example: Cat vs Dog classification



Test Set

- Phase 2: **Test** (Evaluation)

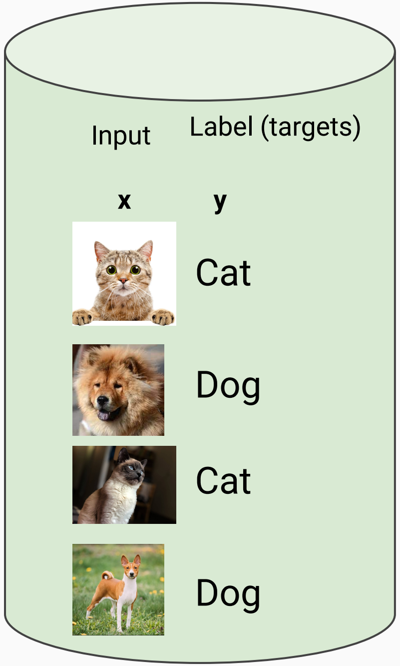




- A good machine learning model should perform well on data never seen before. This ability is called **generalization**.
- The goal of the testing phase (evaluation) is to **measure the performance** on data never seen before (collected in a test set).

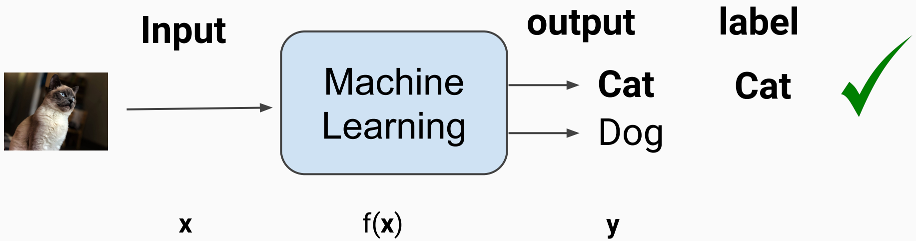
Classification Example

Example: Cat vs Dog classification

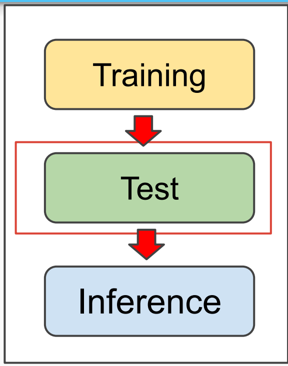


Test Set

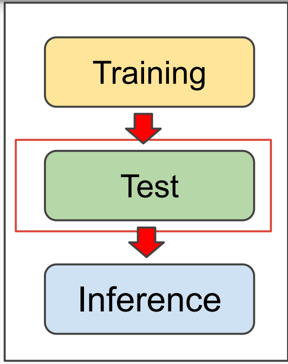
- Phase 2: **Test** (Evaluation)



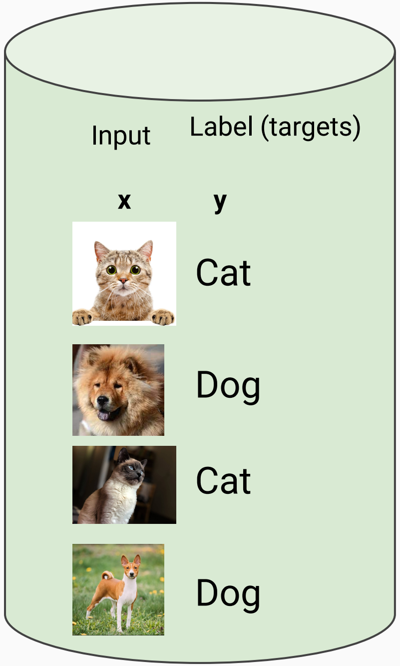
- A good machine learning model should perform well on data never seen before. This ability is called **generalization**.
- The goal of the testing phase (evaluation) is to **measure the performance** on data never seen before (collected in a test set).



Classification Example

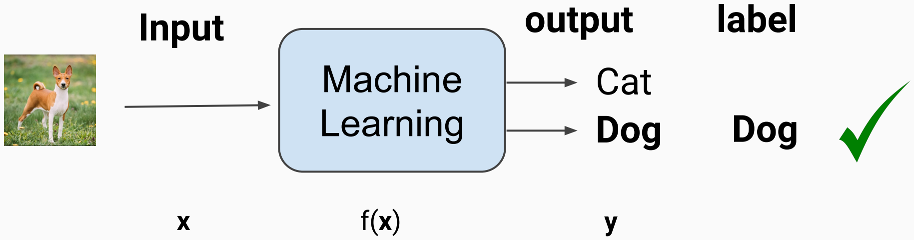


Example: Cat vs Dog classification



Test Set

- Phase 2: **Test** (Evaluation)



- We classified correctly 3 of the 4 entries.
- The **test accuracy** of the systems is: $Acc(\%) = \frac{N_{correct}}{N_{tot}} \cdot 100$

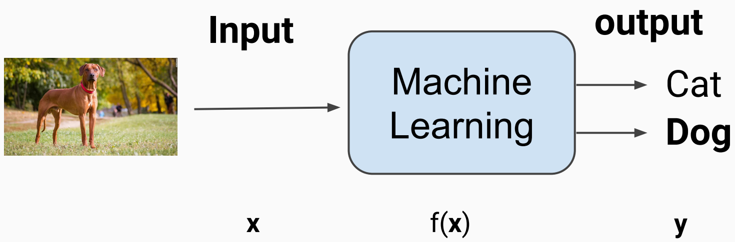
$$Acc(\%) = \frac{3}{4} \cdot 100 = 75\%$$

Classification Example

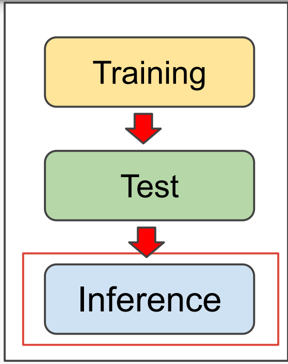
Example: Cat vs Dog classification

Inference Set

- Phase 3: **Inference**

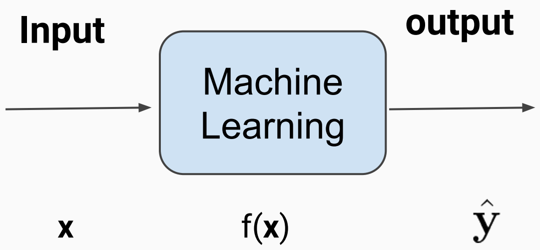


- When we are happy with our machine learning algorithm, we can eventually put it in "production".
- **Inference** is the process of using a trained machine learning algorithm by **running live data points** (without knowing their label).

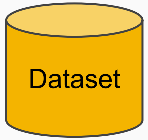


Basic Components:
Datasets

Basic Components

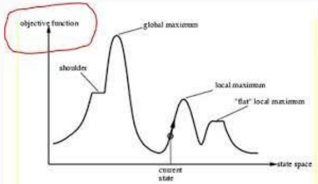


$$\boxed{\hat{\mathbf{y}} = f(\mathbf{x})} \quad \mathbf{x} \in \mathbb{R}^D, \hat{\mathbf{y}} \in \mathbb{R}^K$$

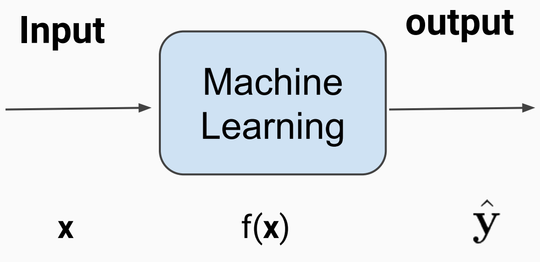


The basic components of a machine learning system are:

- **Datasets**: examples of input-output mappings.
- **Machine learning Model** : implements the function f(x) that maps the inputs into the desired outputs.
- **Objective function**: a measure of “how well” the solution f(x) fits the data.

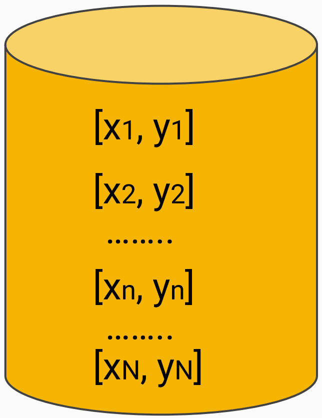


Datasets

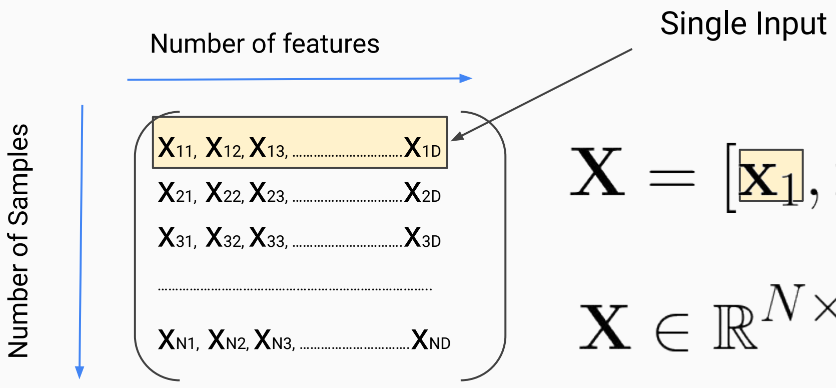


- A dataset is a **collection** of **examples** (sometimes called **data points** or **samples**) containing the desired input-output mappings.

- The inputs **x** are often gathered into a **matrix**:



Dataset



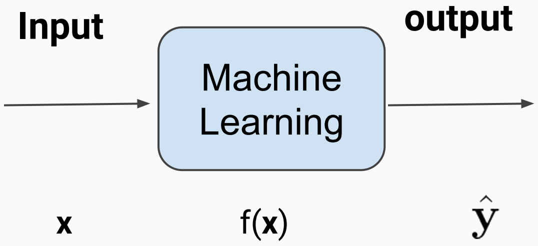
Feature Matrix

$$\mathbf{X} = [\boxed{\mathbf{x}_1}, \mathbf{x}_2, \dots, \mathbf{x}_N]^T$$

$$\mathbf{X} \in \mathbb{R}^{N \times D}$$

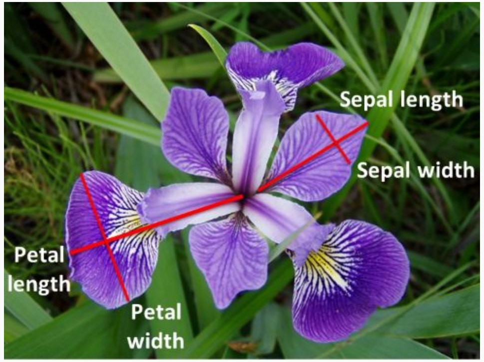
What is a feature?

Features



- A **feature** is a measurable property (attributes) potentially relevant for solving a machine learning problem.
- Each example contains a **feature vector,** which is a collection of some relevant measures.

IRIS classification: 3 classes (*Iris setosa*, *Iris virginica*, *Iris versicolor*)



$$
\begin{array}{cccc}
\text{Sepal Length} & \text{Sepal Width} & \text{Petal Length} & \text{Petal Width} \\
\end{array}
$$

$$
\begin{pmatrix}
5.1 & 3.5 & 1.4 & 0.2 \\
4.9 & 3.0 & 1.4 & 0.2 \\
7.0 & 3.2 & 4.7 & 1.4 \\
\dots & \dots & \dots & \dots \\
5.8 & 3.3 & 6.0 & 2.5
\end{pmatrix}
$$

$$\mathbf{X} \in \mathbb{R}^{150\times 4}$$

Features

- The process of finding proper features for a specific machine learning problem is called **feature extraction**.

f(**x**)

Raw Data → Feature Extraction → **Features** **x** → Machine Learning → **Output** y

Iris setosa
Iris virginica
Iris versicolor

Sepal Length	Sepal Width	Petal Length	Petal Width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
7.0	3.2	4.7	1.4
.....	......	......	
5.8	3.3	6.0	2.5

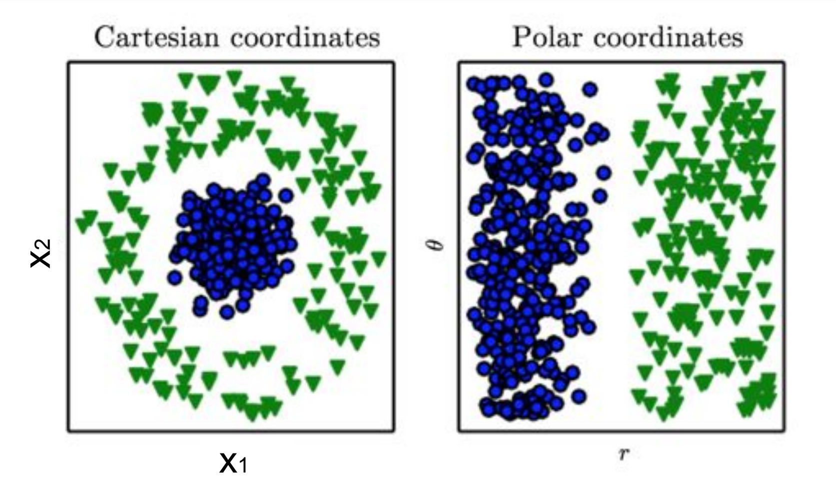
- Manually designing features for a complex task requires a lot of **human time** and **effort**.

- Fortunately, some modern machine learning algorithms (e.g. deep learning) often work well **without** a **feature extraction step** (i.e., we feed the **raw data** such as the pixel of an image into the machine learning system directly).

Features

- Sometimes, we might have too many features and we have to choose only a subset of relevant ones. This process is called **feature selection**.

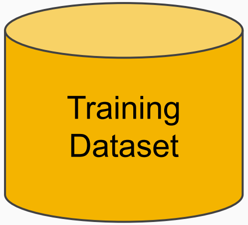
- In some cases, the problem can be solved more easily if we transform the features. This operation is called **feature transformation**.



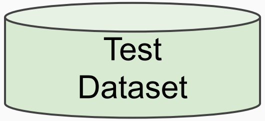
In this case, transforming the features from **cartesian** to **polar coordinate**s helps make the two classes **linearly separable**.

Training and Test Datasets

- In machine learning, we use at least two distinct datasets:



Training Dataset: the set of examples used to find the desired mapping function $f(\mathbf{x})$.



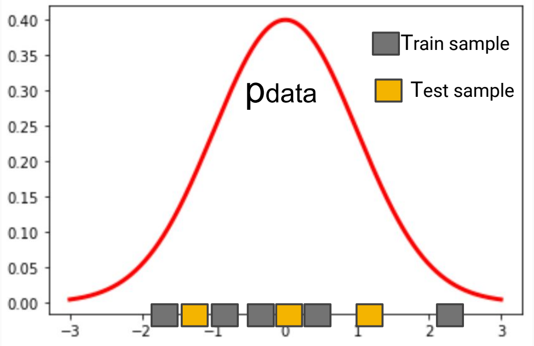
Test Dataset: The set of examples used to assess the performance of the machine learning algorithm (after training).

Training and test sets must contain **different examples** (otherwise, the evaluation would be **biased**).

We might indeed **overestimate** the actual performance of the system if we evaluation samples are already seen during training (just like the performance of a student would be much better if the final exam has the exact same exercises provided as homework).

Training and Test Datasets

- Even though training and test samples are different, a common assumption is that they are sampled from **the same data generation process** p_{data}.



- In this case, for instance, training and test samples are drawn from the same Gaussian distribution.

- **Note**: In real machine learning problems, p_{data} is complex and not accessible. We can only observe samples drawn from it (e.g, images of cats).

- We also assume that each sample is drawn **independently** from any other data point.

- If these conditions hold, the samples are called **independent** and **identically distributed** (*i.i.d*).

Supervised Learning

- **Supervised learning** is **learning from labelled examples**. Training and test examples contain both the input **x** and the desired output y.
- **Classification** and **regression** are supervised learning problems.

$$
\begin{array}{cc}
 & \text{Number of features} \rightarrow \\
\text{Number of Samples} \downarrow &
\begin{pmatrix}
x_{11}, & x_{12}, & x_{13}, & \cdots & x_{1D} \\
x_{21}, & x_{22}, & x_{23}, & \cdots & x_{2D} \\
x_{31}, & x_{32}, & x_{33}, & \cdots & x_{3D} \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
x_{N1}, & x_{N2}, & x_{N3}, & \cdots & x_{ND}
\end{pmatrix}
\end{array}
$$

Feature Matrix

$$\mathbf{X} \in \mathbb{R}^{N \times D}$$

$$
\begin{array}{cc}
 & \text{Number of Outputs} \rightarrow \\
\text{Number of Samples} \downarrow &
\begin{pmatrix}
y_{11}, & y_{12}, & \cdots & y_{1K} \\
y_{21}, & y_{22}, & \cdots & y_{2K} \\
y_{31}, & y_{32}, & \cdots & y_{3K} \\
\cdots & \cdots & \cdots & \cdots \\
y_{N1}, & y_{N2}, & \cdots, & y_{NM}
\end{pmatrix}
\end{array}
$$

Target Matrix (labels or supervision)

$$\mathbf{Y} \in \mathbb{R}^{N \times K}$$

<u>Examples:</u>

- *Linear models*
- *Neural networks*
- *Support vector machine*
- *Naive Bayes*
- *K-nearest neighbor*
- *Random forest*

Unsupervised Learning

- **Supervised learning** is about **learning from observation**. Training and test examples only contain the input x.

Even if we do not have any labels, we can still **observe the data** and hopefully find **useful properties** on their **structure**.

Number of features

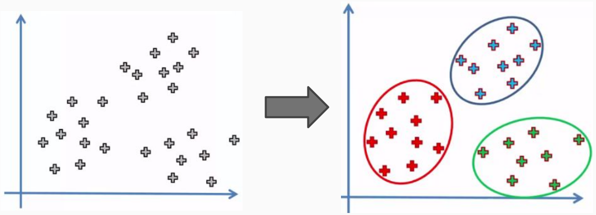
Number of Samples

$$\begin{pmatrix} x_{11}, & x_{12}, & x_{13}, & \dots & x_{1D} \\ x_{21}, & x_{22}, & x_{23}, & \dots & x_{2D} \\ x_{31}, & x_{32}, & x_{33}, & \dots & x_{3D} \\ \dots & & & & \\ x_{N1}, & x_{N2}, & x_{N3}, & \dots & x_{ND} \end{pmatrix}$$

Feature Matrix

$$\mathbf{X} \in \mathbb{R}^{N \times D}$$

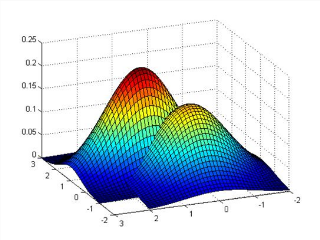
Clustering



We group "close" data into the same cluster.

We cannot give a label to the detected clusters, but likely they contain different types of inputs.

Probability Density Estimation

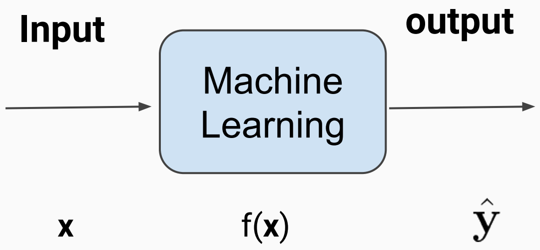


We use training data to estimate pdata. At test time you can say **how likely** is a certain data point.

Basic Components:
Machine Learning Model

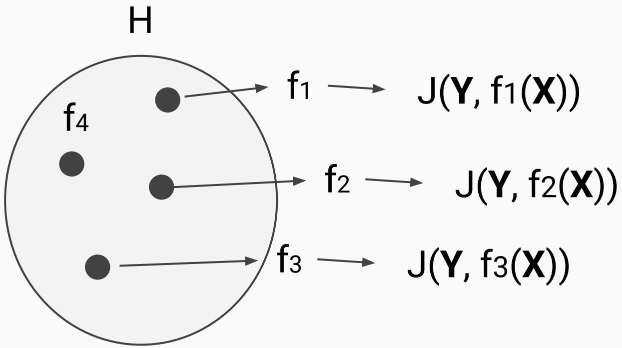
Machine Learning Algorithm

A machine learning algorithm is a **function** f that maps the input into the output.



$$\boxed{\hat{\mathbf{y}} = f(\mathbf{x})} \quad \mathbf{x} \in \mathbb{R}^D, \hat{\mathbf{y}} \in \mathbb{R}^K$$

The set of functions that a machine learning algorithm can implement is the **hypothesis space**.

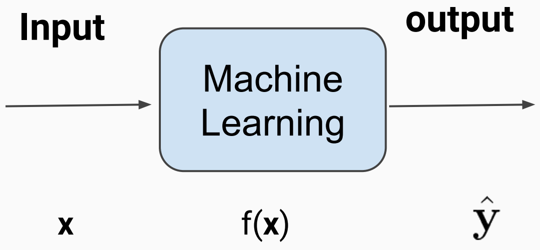


Every function in the hypothesis space is a possible **solution**.

For every solution, we can compute the **objective function** using the training dataset. This tells us "*how good*" is a certain solution.

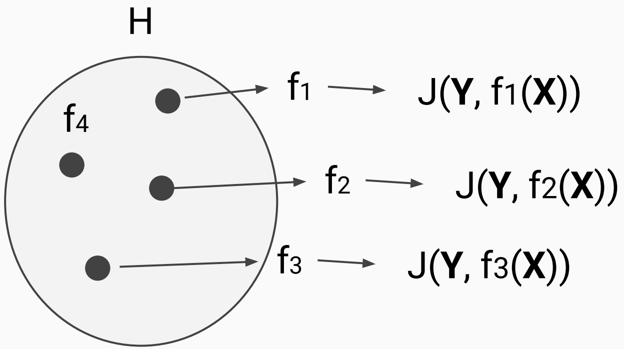
During training, we **explore** the **hypothesis space** until we found a function that **explains well the data**.

Training



$$\boxed{\hat{\mathbf{y}} = f(\mathbf{x})} \quad \mathbf{x} \in \mathbb{R}^D, \hat{\mathbf{y}} \in \mathbb{R}^K$$

Training a machine learning model is about finding a function *f* that explains well the training data:



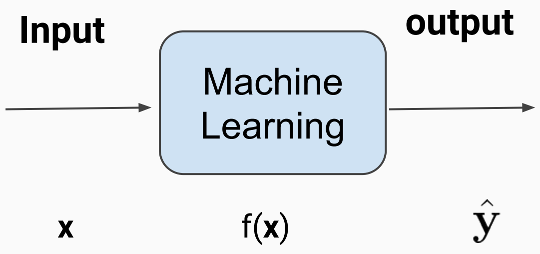
$$\boxed{f^* = \operatorname*{argmin}_f J(\mathbf{Y}, f(\mathbf{X}))}$$

Training a machine learning model requires solving an **optimization problem**.

Basic Components:

Objective Function

Objective



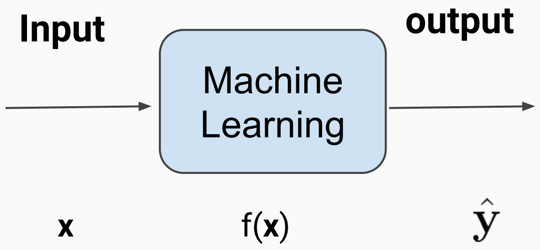
$$\boxed{\hat{\mathbf{y}} = f(\mathbf{x})} \quad \mathbf{x} \in \mathbb{R}^D, \hat{\mathbf{y}} \in \mathbb{R}^K$$

- Training a machine learning model aim to find a function f that “**fits well**” with my training data.
- To quantify “**how good**” are the predictions obtained with the learning function we need to define an **objective function**:

$$\boxed{J(\mathbf{Y}, f(\mathbf{X})) : \mathbb{R}^{N \times K} \to \mathbb{R}}$$

- This function is also called **criterion**.
- By convention, we often want to **minimize** it.

Objective



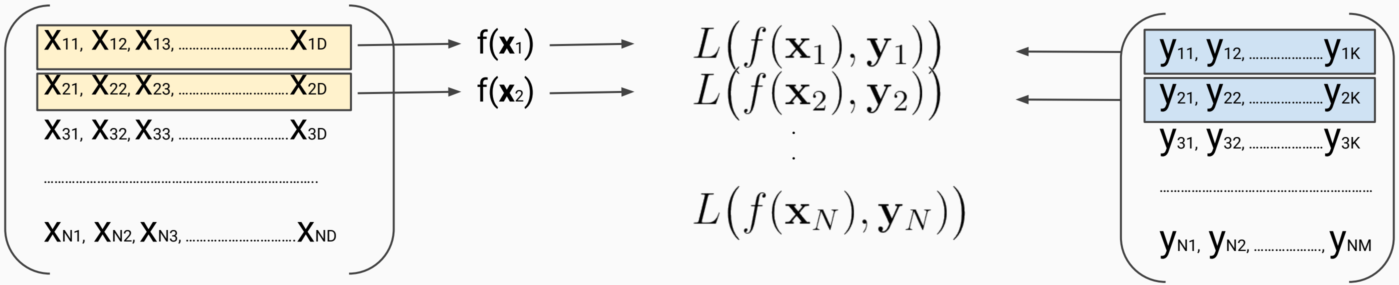
$$\boxed{\hat{\mathbf{y}} = f(\mathbf{x})} \quad \mathbf{x} \in \mathbb{R}^D, \hat{\mathbf{y}} \in \mathbb{R}^K$$

- Often, the objective is written as an average (or sum) over the training samples:

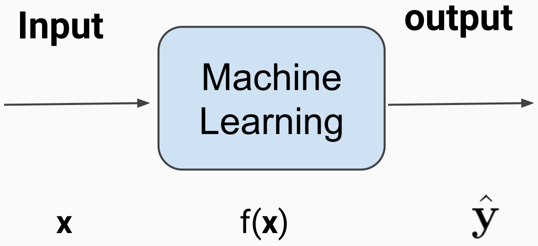
$$\boxed{J(\mathbf{Y}, f(\mathbf{X})) = \frac{1}{N}\sum_{i=1}^{N} L(f(\mathbf{x}_i), \mathbf{y}_i))}$$

The term L is called **Loss**.

The training process based on minimizing such an objective is called **empirical risk minimization**.



Mean Squared Error



Inputs:

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_i, \dots, \mathbf{x}_N]^T$$

Targets:

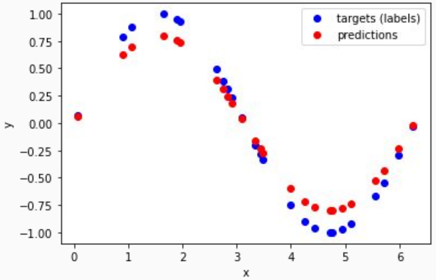
$$\mathbf{y} = [y_1, y_2, \dots, y_i, \dots, y_N]^T$$

$$y_i \in \mathbb{R}$$

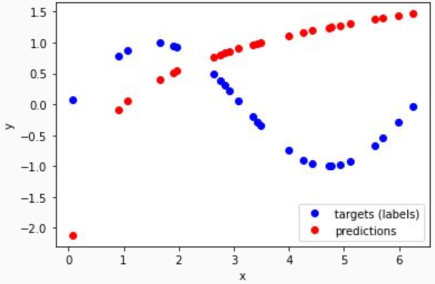
A popular objective used for regression problems is the **Mean Squared Error** (MSE):

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f(\mathbf{x}_i)\right)^2$$

Intuitively, MSE measures **how far** the predictions are from the targeted one.



MSE = 0.018

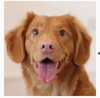


MSE = 2.35

Accuracy

For a classification problem, one possible objective is the **classification accuracy**:

$$\boxed{Acc = \frac{N_{correct}}{N_{tot}}} \qquad Err = 1 - Acc$$



$\mathbf{x}_i$

Machine Learning

p(cat)
p(dog)
p(bird)

p($\mathbf{x}_i$)

Output probabilities

argmax

$$\hat{y}_i = \underset{k}{\operatorname{argmax}}\, P(y_i = k \mid \mathbf{x}_i, \mathbf{w}_k)$$

Label

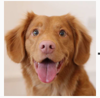
y_i = dog

Prediction

$\hat{y}_i = dog$

Acc

Accuracy



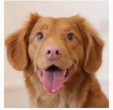
x_i → Machine Learning → p(cat), p(dog), p(bird) → argmax →

Label
y_i = dog

Prediction
$\hat{y}_i = dog$

→ Acc

	cat	**dog**	**bird**	
y = [	0,	1,	2	]^T
	y_1	y_2	y_3	

X = [  ]^T

$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$

Number of Outputs

Number of Samples

$$p(\mathbf{X}) = \begin{pmatrix} \mathbf{0.6} & 0.2 & 0.2 \\ 0.3 & \mathbf{0.5} & 0.2 \\ 0.1 & \mathbf{0.5} & 0.4 \end{pmatrix} \begin{array}{l} p(\mathbf{x}_1) \\ p(\mathbf{x}_2) \\ p(\mathbf{x}_3) \end{array}$$

$$\begin{array}{c} \textbf{cat} \quad \textbf{dog} \quad \textbf{bird} \\ \mathbf{y} = [0, 1, 2]^T \end{array}$$

$$\begin{array}{c} \textbf{cat} \quad \textbf{dog} \quad \textbf{dog} \\ \hat{\mathbf{y}} = [0, 1, 1]^T \end{array}$$

$$Acc = \frac{N_{corr}}{N_{tot}} = \frac{2}{3} = 0.66$$

Accuracy

- Accuracy is a simple metric **easy to interpret** (good for test). However, it is a “**hard**” metric and sometimes machine learning algorithms prefer “**soft**” ones.

- To understand why it is a “hard” metric, let try to consider the following examples:

Example 1

$$p(\mathbf{X}) = \begin{array}{c} \begin{pmatrix} 0.6 & 0.2 & 0.2 \\ 0.3 & \mathbf{0.5} & 0.2 \\ 0.1 & \mathbf{0.5} & 0.4 \end{pmatrix} \end{array} \begin{array}{l} p(\mathbf{x}_1) \\ p(\mathbf{x}_2) \\ p(\mathbf{x}_3) \end{array}$$

$$\mathbf{y} = \overset{\text{cat dog bird}}{[0, 1, 2]^T} \qquad \hat{\mathbf{y}} = \overset{\text{cat dog dog}}{[0, 1, 1]^T}$$

$$Acc = \frac{N_{corr}}{N_{tot}} = \frac{2}{3} = \boxed{0.66}$$

Example 2

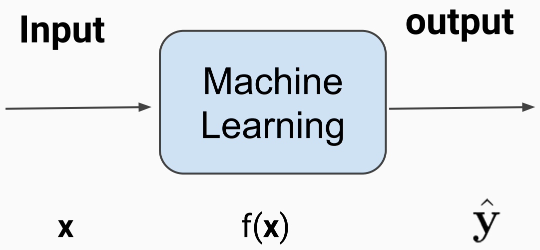
$$p(\mathbf{X}) = \begin{array}{c} \begin{pmatrix} \mathbf{0.9} & 0.1 & 0.0 \\ 0.0 & \mathbf{1.0} & 0.0 \\ 0.1 & \mathbf{0.5} & 0.4 \end{pmatrix} \end{array} \begin{array}{l} p(\mathbf{x}_1) \\ p(\mathbf{x}_2) \\ p(\mathbf{x}_3) \end{array}$$

$$\mathbf{y} = \overset{\text{cat dog bird}}{[0, 1, 2]^T} \qquad \hat{\mathbf{y}} = \overset{\text{cat dog dog}}{[0, 1, 1]^T}$$

$$Acc = \frac{N_{corr}}{N_{tot}} = \frac{2}{3} = \boxed{0.66}$$

- The Accuracy here is **the same**, but the second case looks better because the classifier is more confident about its predictions.

Categorical Cross-Entropy



Inputs:

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_i, ..., \mathbf{x}_N]^T$$

Targets:

$$\mathbf{y} = [y_1, y_2, ..., y_i, ..., y_N]^T$$

$$y_i \in \mathbb{R}$$

- A "softer" alternative is the categorical **cross-entropy**.

$$CCE = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} ln(p_{ik})$$

Also known as
Negative Log-Likelihood (NLL)

Labels

$$\begin{array}{c} \text{cat} \quad \text{dog} \quad \text{bird} \\ \mathbf{y} = [0, 1, 2]^T \end{array} \Rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

One-hot label representation

Example 1

$$p(\mathbf{X}) = \begin{pmatrix} 0.6 & 0.2 & 0.2 \\ 0.3 & 0.5 & 0.2 \\ 0.1 & 0.5 & 0.4 \end{pmatrix} \begin{array}{l} p(\mathbf{x}_1) \\ p(\mathbf{x}_2) \\ p(\mathbf{x}_3) \end{array}$$

Cross Entropy = -⅓ *(
1 * ln(0.6) + 0 * ln(0.2) + 0 * ln(0.2) +
0 * ln(0.3) + 1 * ln(0.5) + 0 * ln(0.2) +
0 * ln(0.1) + 0 * ln(0.5) + 1 * ln(0.4)
)
= **0.70**

Categorical Cross-Entropy

Labels

$$\mathbf{y} = \overset{\text{cat}}{[0}, \overset{\text{dog}}{1}, \overset{\text{bird}}{2]}^T \Rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

One-hot label representation

Example 1

$$p(\mathbf{X}) = \begin{pmatrix} 0.6 & 0.2 & 0.2 \\ 0.3 & 0.5 & 0.2 \\ 0.1 & 0.5 & 0.4 \end{pmatrix} \begin{array}{l} p(\mathbf{x}_1) \\ p(\mathbf{x}_2) \\ p(\mathbf{x}_3) \end{array}$$

Cross Entropy = 0.70

Accuracy = 0.66

Example 2

$$p(\mathbf{X}) = \begin{pmatrix} 0.9 & 0.1 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.1 & 0.5 & 0.4 \end{pmatrix} \begin{array}{l} p(\mathbf{x}_1) \\ p(\mathbf{x}_2) \\ p(\mathbf{x}_3) \end{array}$$

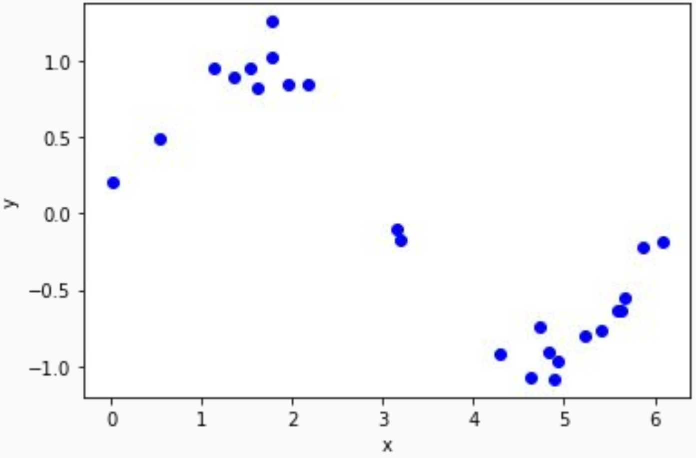
Cross Entropy = 0.34

Accuracy = 0.66

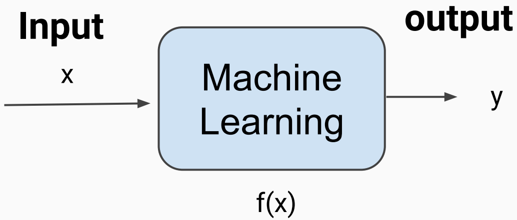
- In this example, the two machine learning models have the same accuracy, but different cross-entropy.
- *Example 2* is better than *Example 1* because the classifier is **more confident** about its predictions. The cross-entropy is thus lower.
- We now have a metric much “softer” that the accuracy.
- The categorical cross-entropy ranges from 0 (**perfect solution**) to +inf (**bad solution**).
- We thus want to **minimize** the metric.

Regression Example

Example 2 : Curve Fitting Problem



Training Set



We have a training set composed of:

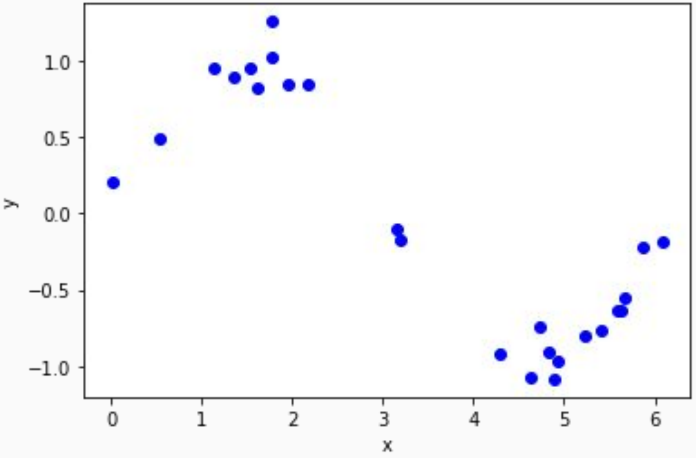
- Inputs $\mathbf{X} = [x_1, x_2, \ldots, x_i, \ldots, x_N]$
- Labels $\mathbf{Y} = [y_1, y_2, \ldots, y_i, \ldots, y_N]$

$x_i, y_i \in \mathbb{R}$

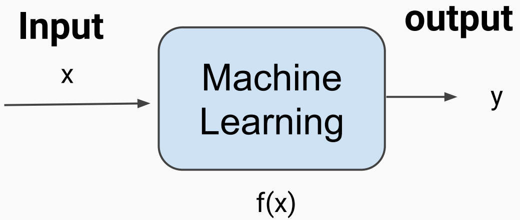
Goal: find a function $f(x) : \mathbb{R} \rightarrow \mathbb{R}$ that fits well with my dataset.

Regression Example

Example 2 : Curve Fitting Problem



Training Set



Naive approach: try all the functions of the hypothesis space and take the one that better explains the training data.

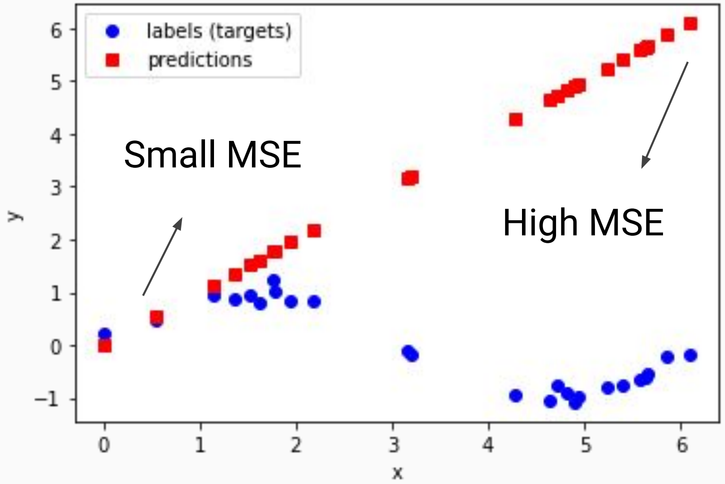
Just for this toy task, let's assume that the hypothesis space is composed of the following functions:

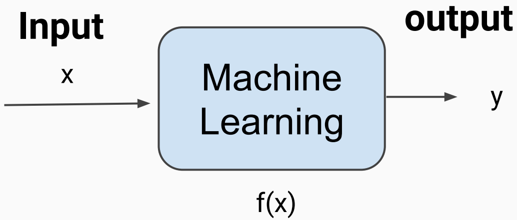
Candidate 1: $f(x) = x$
Candidate 2: $f(x) = e^x$
Candidate 3: $f(x) = \sin(x)$
Candidate 4: $f(x) = \cos(x)$

How well do they fit the training dataset?

Regression Example

Candidate 1 : $f(x) = x$





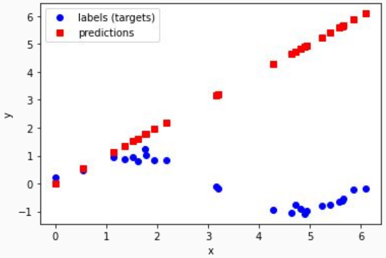
We can use the **Mean Squared Error** (MSE) as an objective:

$$MSE = \frac{1}{N} \sum_{i=0}^{N} (y_i - f(x_i))^2$$

- MSE is "*low*" when the predictions approach the targets and "*high*" when the predictions are far away from the targets.

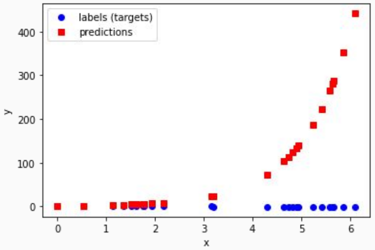
Regression Example

Candidate 1 : $f(x) = x$



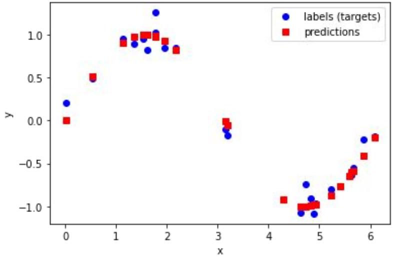
MSE = 19.49

Candidate 2 : $f(x) = e^x$



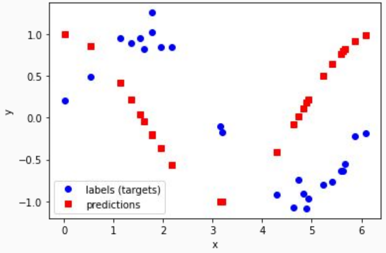
MSE = 28891.95

Candidate 3 : $f(x) = \sin(x)$



MSE = 0.013

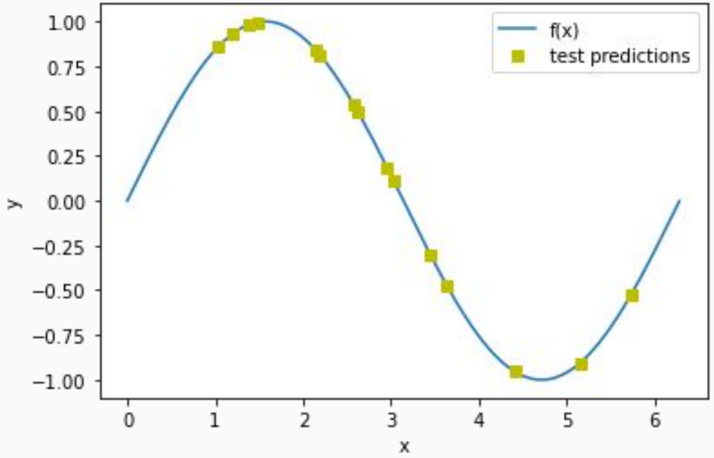
Candidate 4 : $f(x) = \cos(x)$



MSE = 1.18

Regression Example

- Our function f(x) = sin(x) **fits well** with the **training data**.
- However, we are more interested to see how well this mapping works on data never seen before (**generalization**).
- Let's thus compute the MSE on the **test set**:



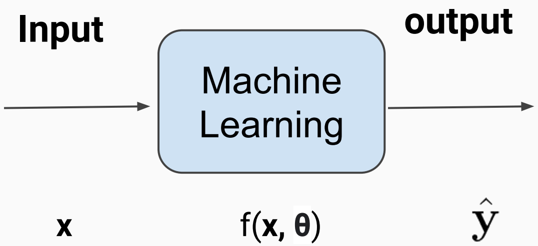
MSE = 0.010

- The function discovered during training fits well the test data: the **mean square error is close to zero**.
- This evidence suggests that we learned a function that **generalizes** well on new data points.

Parametric Models

Parameters

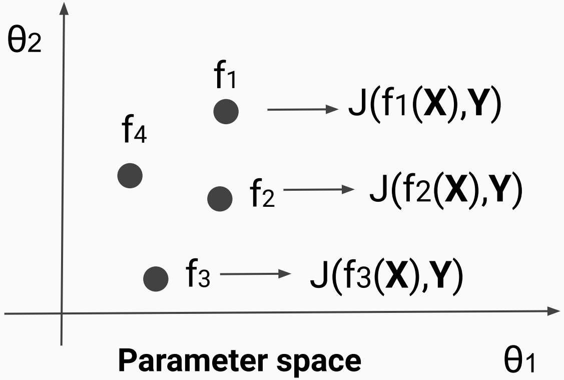
The function implemented by a machine learning often depends on some **parameters** θ



$$\boxed{\hat{\mathbf{y}} = f(\mathbf{x}, \boldsymbol{\theta})} \quad \mathbf{x} \in \mathbb{R}^D, \hat{\mathbf{y}} \in \mathbb{R}^K \qquad f : \mathbb{R}^D \rightarrow \mathbb{R}^K$$

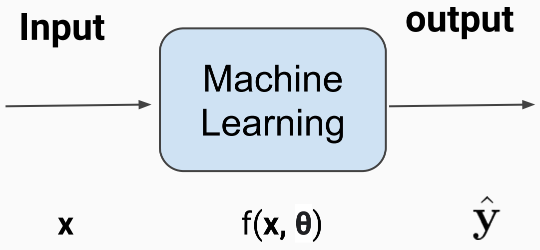
$$\boldsymbol{\theta} = [\theta_1, \ldots, \theta_P]^T \quad \boldsymbol{\theta} \in \mathbb{R}^P$$

- For each **parameter** configuration, the machine learning model implements a **different function**.



- In this case, the hypothesis space is equivalent to the **parameter space**.
- For every point of the parameter space, we can compute the **objective function** using the training dataset.
- During training, we **explore** the **parameter space** until we found a function that explains well the data.

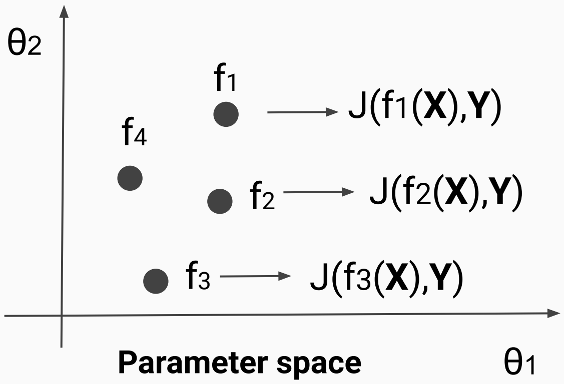
Parameters



$$\boxed{\hat{\mathbf{y}} = f(\mathbf{x}, \boldsymbol{\theta})} \quad \mathbf{x} \in \mathbb{R}^D, \hat{\mathbf{y}} \in \mathbb{R}^K \qquad f : \mathbb{R}^D \to \mathbb{R}^K$$

$$\boldsymbol{\theta} = [\theta_1, ..., \theta_P]^T \quad \boldsymbol{\theta} \in \mathbb{R}^P$$

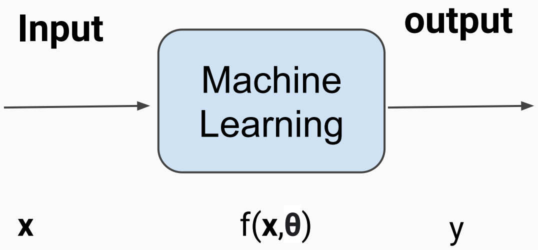
For parametric machine learning, the training is about solving the following optimization problem:



$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J\big(\mathbf{Y}, f(\mathbf{X}, \boldsymbol{\theta})\big)$$

Linear Regression

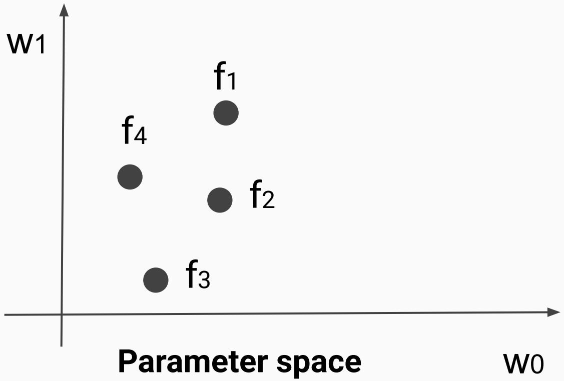
For instance, our machine learning model can implement **linear functions**

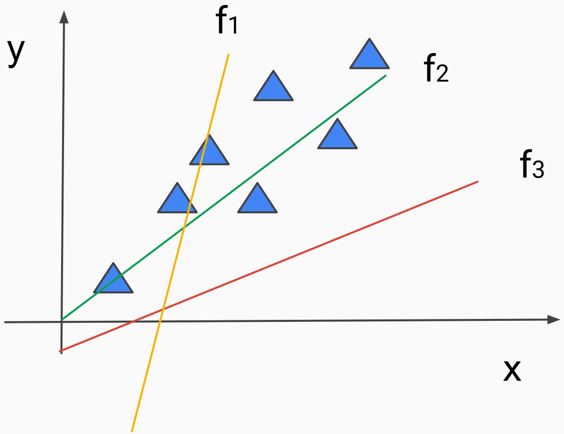


$$f(x, \mathbf{w}) = w_0 + w_1 x$$

$$\boldsymbol{\theta} = \mathbf{w} = [w_0, w_1]$$

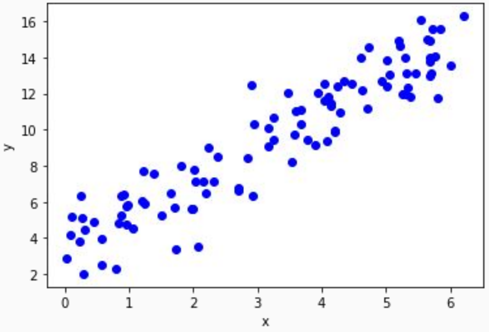
Depending on the values of w_0 and w_1, we implement different linear functions.

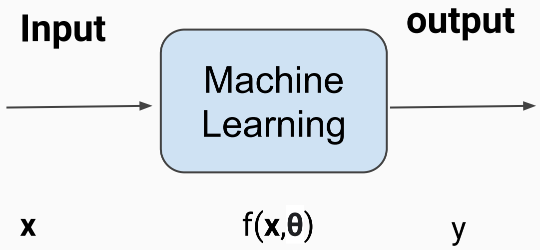


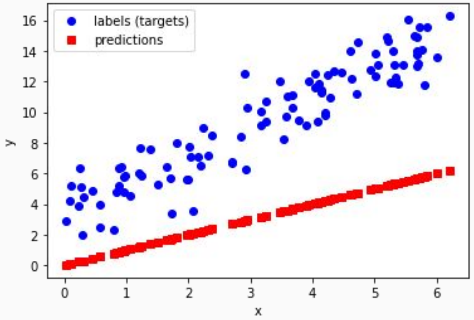


The goal of training is to find the parameter configuration that explains well the training data (triangle points).

Linear Regression







We can start with **random parameters** and evaluate how the corresponding function performs on the training set.

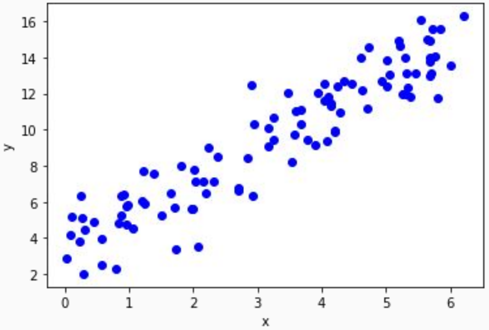
Let's start for instance with $w_0=0$ and $w_1=1$.

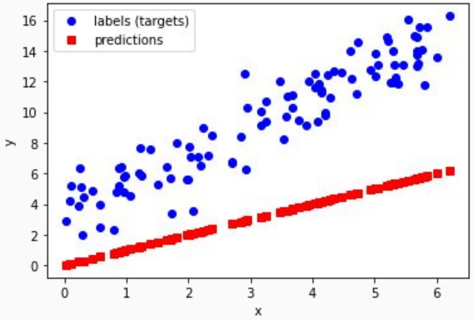
MSE = 40.60

$w_0=0$, $w_1=1$

That's pretty high. The current function does not explain well the training data.

Linear Regression





MSE = 40.60

w_0=0, w_1=1

$$f(x, \mathbf{w}) = w_0 + w_1 x$$

- Let's now try to do a little step forward and backward for all the parameters, and let's monitor how the performance changes:

$$MSE(\mathbf{Y}, f(\mathbf{X}, \boxed{w_0{+}0.05}, \boxed{w_1{+}0.05})) = \boxed{19.28}$$

$$MSE(\mathbf{Y}, f(\mathbf{X}, w_0{+}0.05, w_1{-}0.05)) = 62.17$$

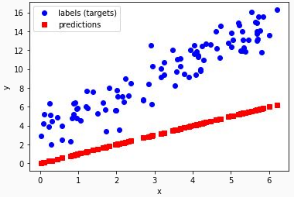
$$MSE(\mathbf{Y}, f(\mathbf{X}, w_0{-}0.05, w_1{+}0.05)) = 28.52$$

$$MSE(\mathbf{Y}, f(\mathbf{X}, w_0{-}0.05, w_1{-}0.05)) = 77.83$$

Ok, the best MSE is observed when increasing a bit both the parameters.
Let's do a step in this direction!

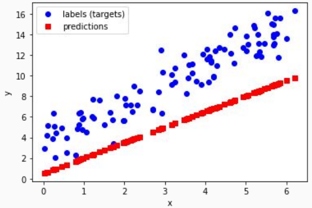
Linear Regression

- We now have a slightly better function, but we are still unhappy.
- To further improve it, we can repeat this game **multiple times**.



MSE = 40.60
w_0=0, w_1=1

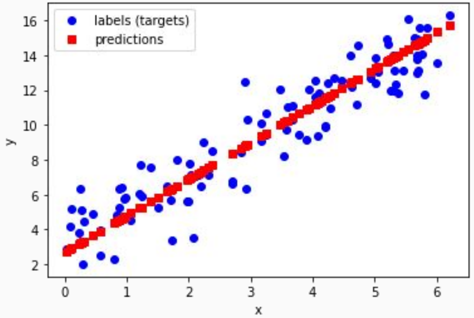
Epoch 0



MSE = 19.28
w_0=0.05, w_1=1.05

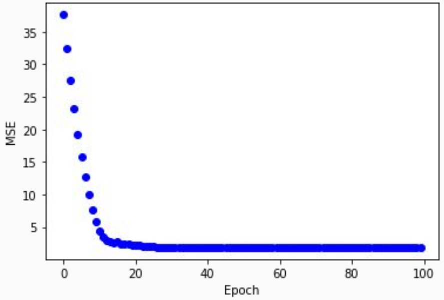
Epoch 1

……………



MSE = 1.92

w_0=2.7, w_1=2.1



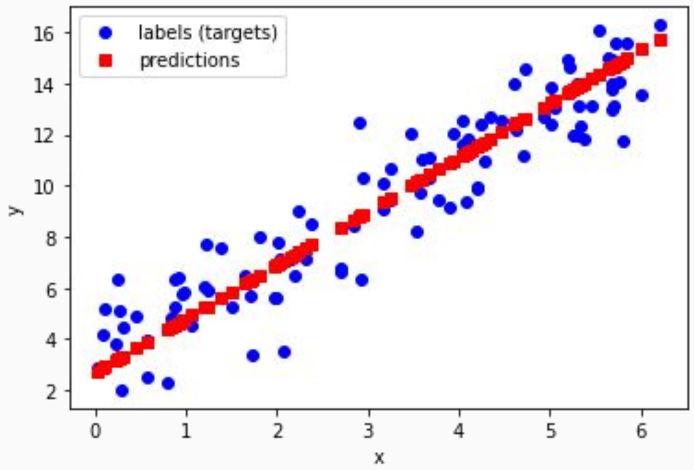
- The MSE decreases fast in the first epochs and then converges to a value close to 0.

Linear Regression

We found a function that matches reasonably well with the **training data**.

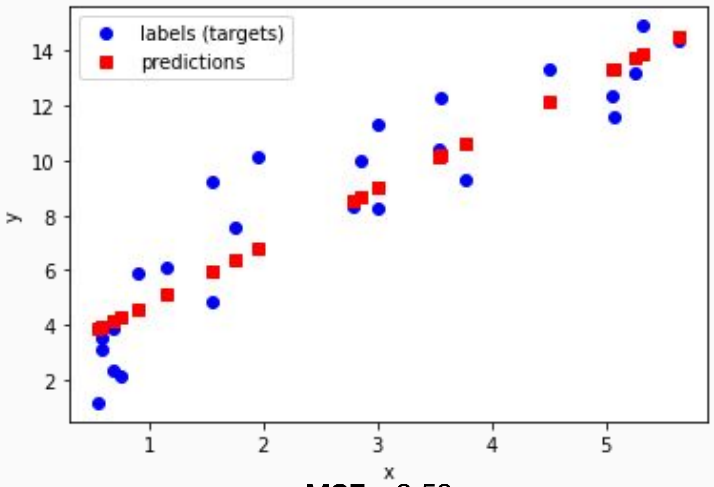
But what about the performance of the **test set**?

Training



MSE = 1.92

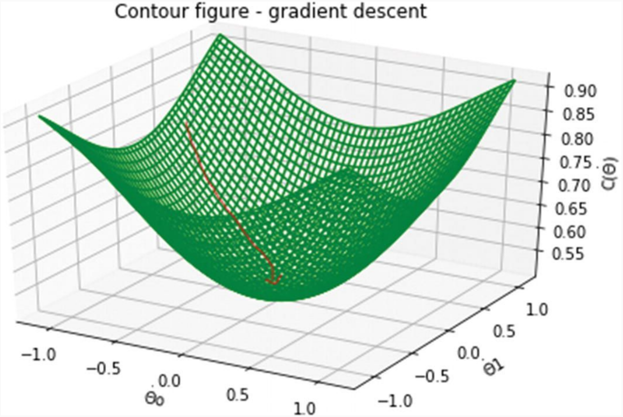
Test



MSE = 2.52

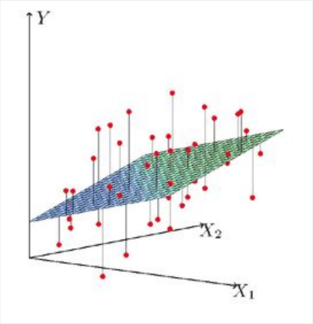
Linear Regression

- We have seen a simple way to train a parametrized linear regressor.
- We trained it by applying **small variations** to our parameters and choosing the parameter configuration that maximized the MSE.
- As we will see in future lectures, this "**little step**" in the direction that optimizes the cost function is what we will call ***gradient***.



Linear Regression in High Dimension

When the number of inputs is higher than 1 we can write the **linear function and MSE objective as follows**



Model:

$$f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{x}^T \boldsymbol{w} + \boldsymbol{b}$$

Objective:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - f(\mathbf{x}_i)\right)^2$$

$$\min_{w} ||Xw - y||_2^2$$

Number of features

Number of Samples

$$\begin{pmatrix} x_{11}, x_{12}, x_{13}, \ldots\ldots x_{1D} \\ x_{21}, x_{22}, x_{23}, \ldots\ldots x_{2D} \\ x_{31}, x_{32}, x_{33}, \ldots\ldots x_{3D} \\ \ldots\ldots\ldots\ldots \\ x_{N1}, x_{N2}, x_{N3}, \ldots\ldots x_{ND} \end{pmatrix}$$

Feature Matrix

$$\mathbf{X} \in \mathbb{R}^{N \times D}$$

Number of Samples

$$\begin{pmatrix} y_1, \\ y_2 \\ y_3 \\ \ldots \\ y_N \end{pmatrix}$$

Target

An Interlude: Sklearn

Defacto standard for applying non-deep learning ML methods:

- Simplified interface to all regression and classification model
    - Object (e.g. Myclassifier stores all key info e.g. model parameters)
    - MyClassifier.fit(Train data)
    - MyClassifier.predict(test data)
- Lots of useful functions
    - Loading toy datasets
    - Splitting up data into train and test set
    - Evaluating different metrics using the prediction (e.g. accuracy)

Capacity

- Different machine learning algorithms have different **hypothesis spaces**.

- The **capacity** (also called representational capacity) of a machine learning model attempts to quantify how “*big*” (or “*rich*”) is the model. In other words how many relationships can it quantify

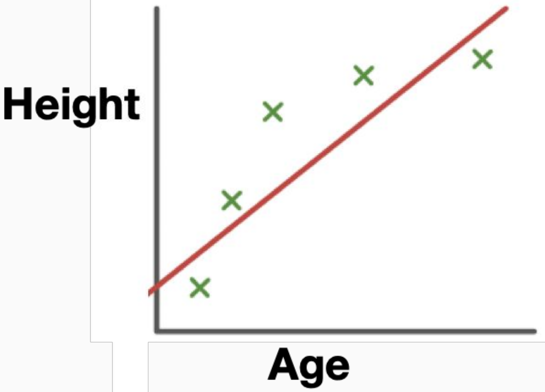
- It is usually not intended as the number of functions in the hypothesis space (that can easily be infinite also for simple machine learning models).

- Instead, it more often refers to the **variability** in terms of “**family**” of **functions** (*expressive power*, *richness*).

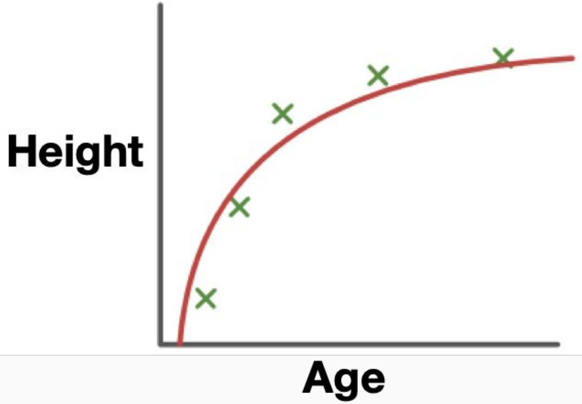
- For instance, a complex machine learning model that can implement *linear*, *exponential*, *sinusoidal*, and *logarithmic* functions have a larger capacity than one that implements *linear* functions only.

Capacity Example

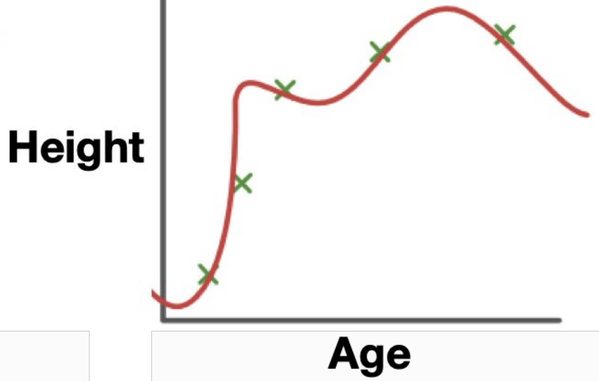
- In *some* cases (especially for a fixed family of models like polynomials) capacity can be related directly to the number of parameters



$$b * x + a$$



$$c * x^2 + b * x + a$$



$$e * x^4 + d * x^3 + c * x^2 + b * x + a$$

Increased capacity

Generalization

- Training a machine learning model often requires solving an **optimization problem**.

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J\big(\mathbf{Y}, f(\mathbf{X}, \boldsymbol{\theta})\big)$$

We have to find the parameters of the function f that minimizes the objective function using the **training data**.

- However, we are more interested in the performance achieved on data **never seen before**.
- **Generalization** is the ability of a machine learning algorithm to perform well to new, previously unseen data.
- A machine learning model generalizes well if the test loss is low enough (according to our application)

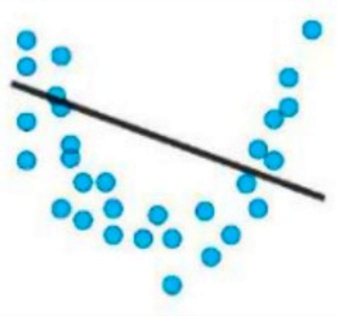
Training Loss: Objective function computed with the training set.

Test Loss: Objective function computed with the test set.

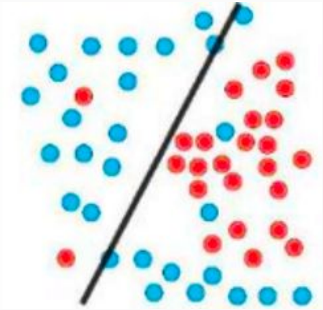
Underfitting

- By analyzing training and test losses, we can identify some “**pathologies**” that often affect machine learning models.

- One of these conditions is called **underfitting.**
- It happens when the model is not able to achieve a **training loss** sufficiently **low**.



Regression



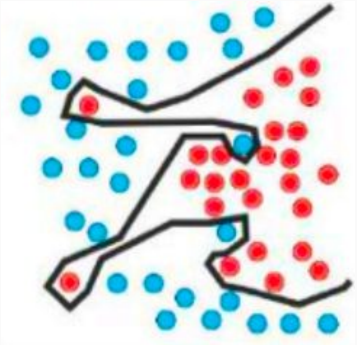
Classification

- In both cases, the function found by the machine learning algorithm is **too simple** to explain well the training data.

Overfitting

- Another possible pathology is **overfitting**.
- It happens when the **gap between the training and test losses** is too **large**.

Regression



Classification

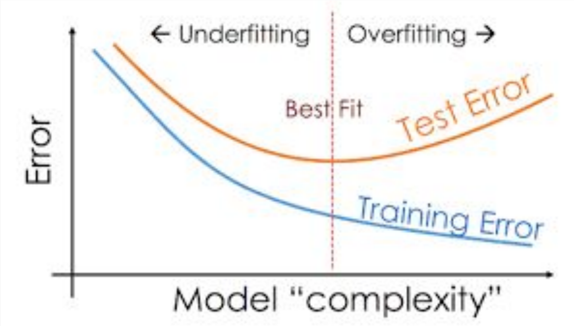
- In both cases, the function found by the machine learning algorithm is **too complex** to explain well the training data.
- In an extreme case, the model just stores the training samples and thus behaves as a **memory without generalization capabilities**.

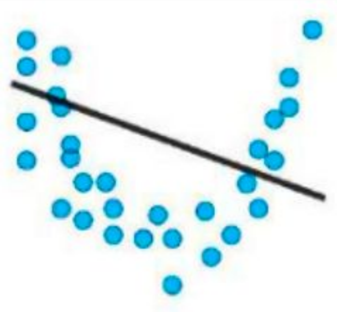
Underfitting, Overfitting, Capacity

- Underfitting and Overfitting are connected to the **capacity** of the model.

- <u>Intuitively:</u>

Too Low Capacity → Underfitting
Proper Capacity → Proper Fitting
Too High Capacity → Overfitting





Undefitting
(*low capacity*)

Proper fitting
(*proper capacity*)

Overfitting
(*low capacity*)

For each task, we have to choose a model with **proper complexity**.

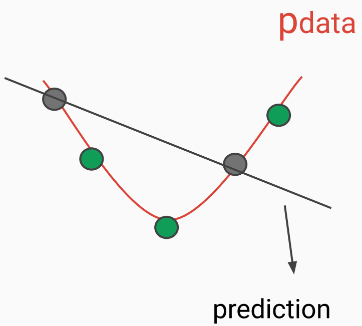
Underfitting, Overfitting, Dataset size

- Underfitting and Overfitting are influenced by the **number of training samples** as well.

- <u>Intuitively:</u>

More Training Examples ➡	Better Generalization
Few Training Examples ➡	Overfitting

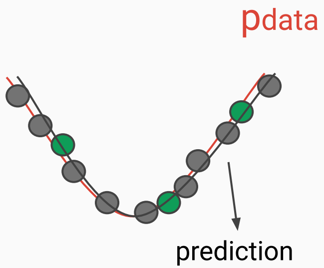
<u>Small Number of Training Samples</u>



In this case, the **training loss** is **low**.

We are anyway in a **overfitting** regime because the **test loss** is **high.**

<u>High number of Training Samples</u>

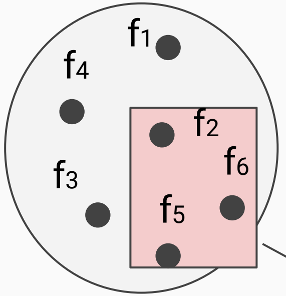


Also in this case the **training loss** is **low**.

However, this time we have a good generalization because **test loss** is **low.**

Regularization

- **Regularization** techniques that aim to counteract overfitting (and thus improve generalization).
- Different techniques have been proposed in the machine learning literature (e.g., L1 regularization, L2 regularization).
- A common aspect is that they embed some kind of **preference for some solutions** over others (using **prior knowledge**).



Hypothesis space

- A way to improve generalization is to **penalize complexity**.

Occam's razor (1287-1346)

Among competing hypothesis that explain the training data equally well, we should choose the "simplest" one

This selection is based on prior knowledge on how could be a proper function for my problem.

Regularization Empirical Risk Minimization

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{Y}, f(\boldsymbol{X}, \boldsymbol{\theta})) + \alpha\Omega(\boldsymbol{\theta})$$

Example: L2 regularization

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{Y}, f(\boldsymbol{X}, \boldsymbol{\theta})) + \alpha\|\boldsymbol{\theta}\|_2$$

Sklearn L2 Regression

https://scikit-learn.org/stable/modules/linear_model.html#ridge-regression-and-classification

$$\min_{w} ||Xw - y||_2^2 + \alpha ||w||_2^2$$

```
>>> from sklearn import linear_model
>>> reg = linear_model.Ridge(alpha=.5)
>>> reg.fit([[0, 0], [0, 0], [1, 1]], [0, .1, 1])
Ridge(alpha=0.5)
>>> reg.coef_
array([0.34545455, 0.34545455])
>>> reg.intercept_
0.13636...
```

Hyperparameters and Validation Set

Hyperparameters

- We have seen that in several machine learning models we have to **learn** the **parameters θ** that implement the desired input-output mapping (e.g., the slope w_1 and the intercept w_0 of our linear model)

- There are special “parameters” to set to properly **control the learning algorithm itself**.

- These “special parameters” are called **hyperparameters**.

- One example of a hyperparameter is the α in L2 Regularized models

$$\min_{w} ||Xw - y||_2^2 + \alpha ||w||_2^2$$

- We cannot compute the gradient for these variables and users have to set them manually.

Hyperparameters

How do we choose the hyperparameters?

- One way is to perform **several training experiments** with different sets of hyperparameters and choose the **best one**.

To select the best one **we cannot use** the performance achieved on the **training set**.

Why? Because we increase the risk of overfitting.

To select the best one **we cannot use** the performance achieved on the **test set**.

Why? Because we will overestimate the actual performance of the system.

Validation Set

- We can employ a third set, called **validation set**, to choose the best set of hyperparameters.

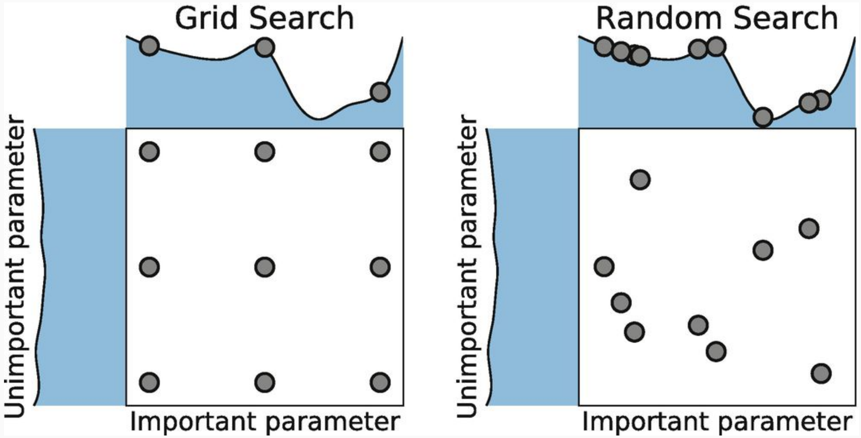
- The validation set is normally extracted from the training set (e.g, 10%-20% of the training data are devoted to validation).

- The training set is used to find the best parameters, the validation set is use only to select the hyperparameters.

Hyperparameter Search

- Searching for the best hyperparameter is usually **expensive** in real machine learning problems because we have to **train the model multiple times** before finding the best configuration.
- One possible way is to initialize the hyperparameters with "reasonable" values (based on our experience or default setting suggested in the literature).
- Then, we can fine-tune the initial configuration through a **hyperparameter search**:

1. *Manual search*
2. *Grid Search*
3. *Random Search*
4. *Bayesian Optimization*



Hyperparameters

Parameters

- They are part of the model f(**x**,**θ**).
- They are estimated during **training** using a **training set**.
- In gradient descend, we train the model by computing a gradient of the objective over the parameters.
- Examples of parameters are the weights w_0 and w_1.

Hyperparameters

- They are external to the model f(**x**,**θ**).
- They are not estimated during training, but during the **hyperparameter search** (performed on the **validation set**)
- We cannot compute the gradient over the hyperparameters.
- Examples are the learning rate, batch size, number of epochs.