# Fundamentals of Information Systems

## Python Programming (for Data Science)

### Master’s Degree in Data Science

Gabriele Tolomei

gtolomei@math.unipd.it
University of Padua, Italy
2018/2019
December, 3 2018

## Lecture 11: A Machine Learning Primer

## What is Machine Learning?

- There exist several definitions of *what* is **Machine Learning** (**ML**):

  1. "A computer program is said to **learn from experience** *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*." (Tom Mitchell)

  2. "ML is an application of artificial intelligence (AI) that provides systems the ability to **automatically learn** and improve from experience (i.e., *observed data*) **without being explicitly programmed**" (source).

  ...

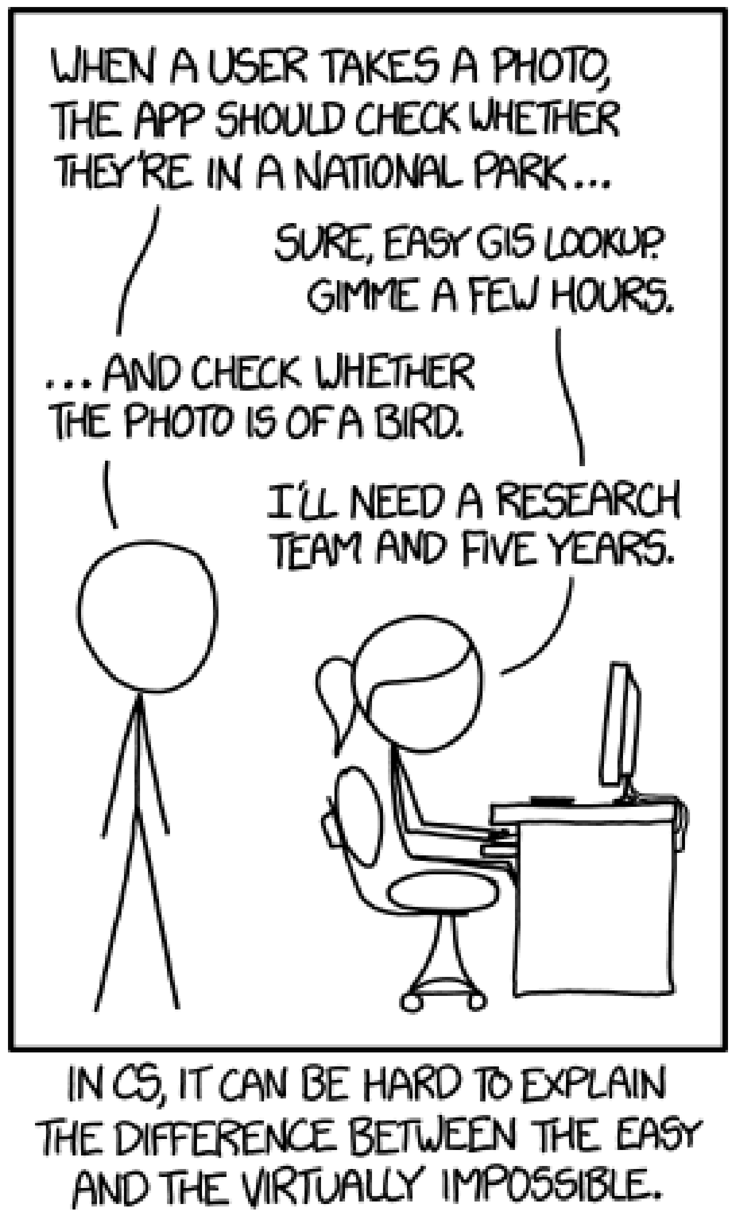
## Just Take a Step Back

- Traditionally, we design computer programs to solve specific tasks which would otherwise require a huge human effort.
- For example, consider the task of creating a vocabulary of words from a collection of text documents.
- To do so, we need to be able to build an **algorithm** which solves the task we aim to perform.
- In other words, we have to *mathematically encode* our task into a **computable function** which takes some **input** (e.g., a collection of text documents) and returns some **output** (e.g., a vocabulary of words).

## Different Types of Tasks

- Some tasks, however, simply **cannot** be pinpointed down mathematically, as the function they encode to is just too hard to be implemented in a traditional computer program.
- Consider how would you possibly write a computer program, which given as input an image gets as output a boolean answer that tells whether the image contains or not a bird...
- **Idea**: Access to a collection of, say, 1M images each one with a **label** associated (e.g., `bird`/`no_bird`) and output a computer program that detects birds in images.

## xkcd Tells Us Better!


WHEN A USER TAKES A PHOTO, THE APP SHOULD CHECK WHETHER THEY'RE IN A NATIONAL PARK...
SURE, EASY GIS LOOKUP. GIMME A FEW HOURS.
...AND CHECK WHETHER THE PHOTO IS OF A BIRD.
I'LL NEED A RESEARCH TEAM AND FIVE YEARS.
IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

(Image Source)

## Taxonomy of ML

- Roughly speaking there are two main **categories** of ML tasks:

    1. **Supervised Learning**: Given a set of **labeled** examples, **predict** the labels of *new and unseen* examples
    2. **Unsupervised Learning**: Given a set of examples, **find structure** in the data (e.g., *clusters*, *subspaces*, *manifolds*)

## Supervised Learning

- We can further divide supervised learning into two sub-categories, depending on the **label** we want to predict.
    - **Regression**: if the label takes on *continuous* values
    - **Classification**: if the label takes on *discrete* values

## Supervised Learning: Regression vs. Classification

- **Regression**:
    - value of a house
    - probability of click on a link to a web page
- **Classification**:
    - spam *vs.* non-spam emails (**binary classification**)
    - movie ratings on a 1 to 5 scale (**k-ary or multiclass classification**)

## The General (Supervised) ML Pipeline

0) Be sure your problem needs *actually* to be tackled using ML!

1) **Data Collection**: Get **labeled data** from the domain of your interest (may be the hardest part!)

2) **Feature Engineering**: Represent your data with a "machine-friendly" format

3) **Model Training**: Build one (or more) **learning models** (more on this later)

4) **Model Selection/Evaluation**: Pick the best-performing model according to some **quality metrics**

## 1. Data Collection

## Motivation

- Any ML technique requires data to operate on!
- Supervised Learning, in particular, needs **labeled data**, which may be even harder to get (e.g., a set of emails with `spam`/`non_spam` tags).
- Most of the time, this step involves combining multiple and possibly heterogeneous data sources.

## 2. Feature Engineering

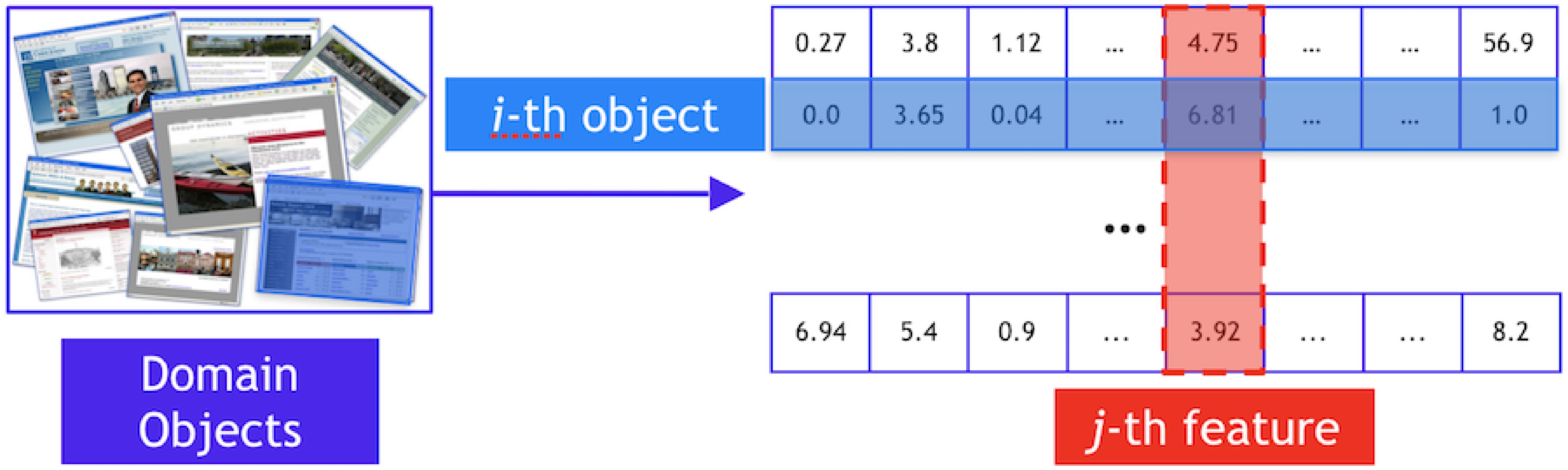
## Motivation

- Collected data must be encoded to a **machine-readable** format.
- Each object of our problem domain is transformed into an $n$-dimensional **feature vector**.
- Each **feature** captures a specific property of the object we believe it is useful to "separate" different objects.
- In the following, we refer to **instance** (or **example**) to identify the feature vector of an object **and** its associated label.

## Characteristics of Features

- Feature values can be either **categorical** (e.g., *gender*) or **continuous** (e.g., *weight*).
- A feature can be derived *locally* from a single object (e.g., the *annual salary* of a person)
- Or it can result from more complex computation on the whole data collection (e.g., the *tf-idf* score of a word in a document computed against a corpus of documents).

## Example: *spam* vs. *non-spam* emails



## Challenges and Solutions

- **Problem**: Collected data is far from being perfect!
- Many challenges need to be addressed before moving down to next stages of the ML pipeline.
- Overall, this is accomplished with an intermediate **Data Preprocessing** step.

## Handling Missing Values

| Challenge | Description | Solution |
|---|---|---|
| Missing values | A feature value may not be available for one or more instances | Replace missing values with the **median** (*continuous*) or the **mode** (*categorical*) of the existing values |

## Managing Data Sparsity

| Challenge | Description | Solution |
|---|---|---|
| Sparsity | Most of the instances contain just a small subset of the features | Use “sparse-friendly” data structures (e.g., DOK) |

## Handling Outliers

| Challenge | Description | Solution |
|---|---|---|
| Outliers | One or more instances have out-of-range values for one or more features | Retention vs. Exclusion (*trimming* or *Winsorising*) |

## Mixing Categorical and Continuous Feature Values

| Challenge | Description | Solution |
|---|---|---|
| Mix of continuous and discrete values | Feature set contains both numerical and categorical values | Transform categorical features using *one-hot encoding* |

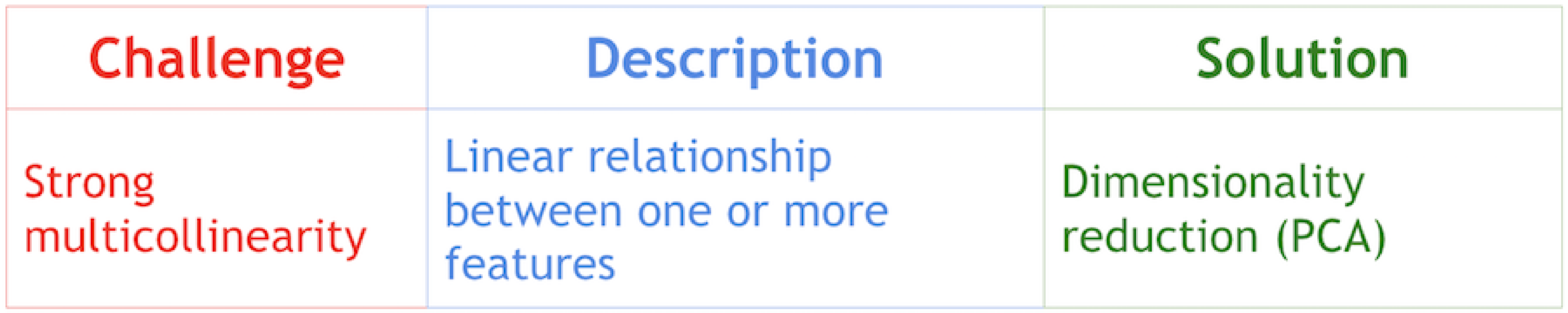
## Different Feature Scales

| Challenge | Description | Solution |
|---|---|---|
| Multiple feature magnitudes | Feature set contains very wide range of values | Standardization (min-max, z-scores) |

## Managing Imbalance

| Challenge | Description | Solution |
|---|---|---|
| Class imbalance | Instances labeled with the class of interest represents a tiny fraction of the total | Over-/Under-sampling, cost-sensitive learning |

## Relationship between Features

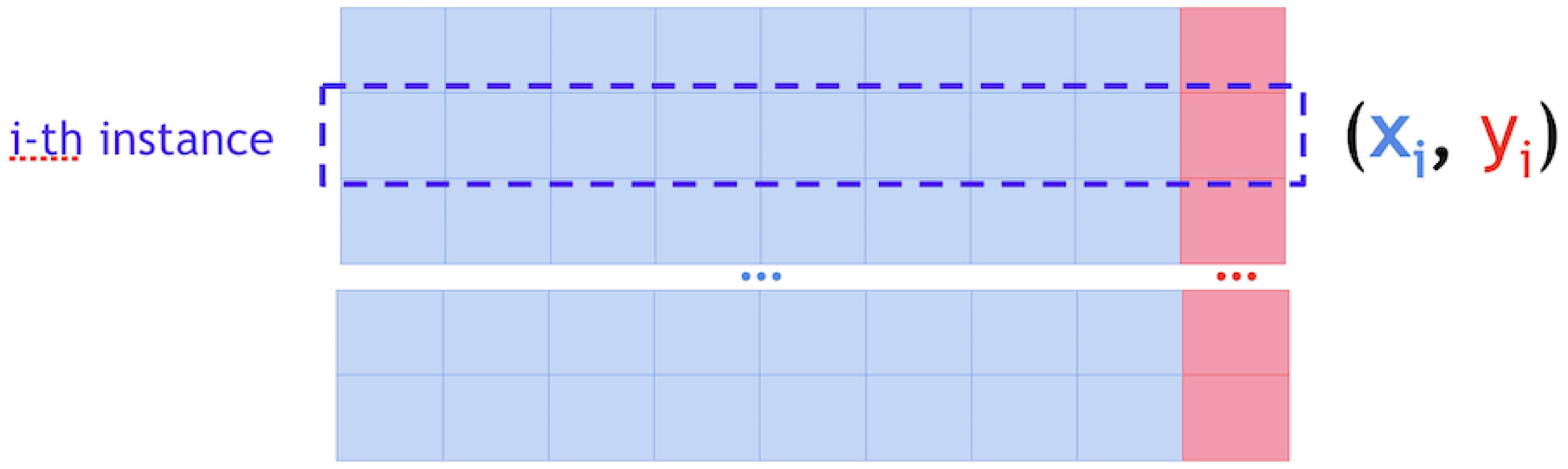


| Challenge | Description | Solution |
|---|---|---|
| Strong multicollinearity | Linear relationship between one or more features | Dimensionality reduction (PCA) |

## A Bit of Notation

- We denote by $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$ our resulting labeled dataset.
- This is made of $m$ **instances**; the $i$-th instance is represented by the tuple $(\mathbf{x}_i, y_i)$, where:
  - $\mathbf{x}_i \in \mathbb{R}^n$ is an $n$-dimensional **feature vector**, i.e., $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,n})^T$
  - $y_i \in \mathbb{R}$ if the label is **continuous** (**regression**) or $y_i \in \{1, 2, \ldots, k\}$ if the label is **categorical** and takes on $k$ values (**k-ary classification**)

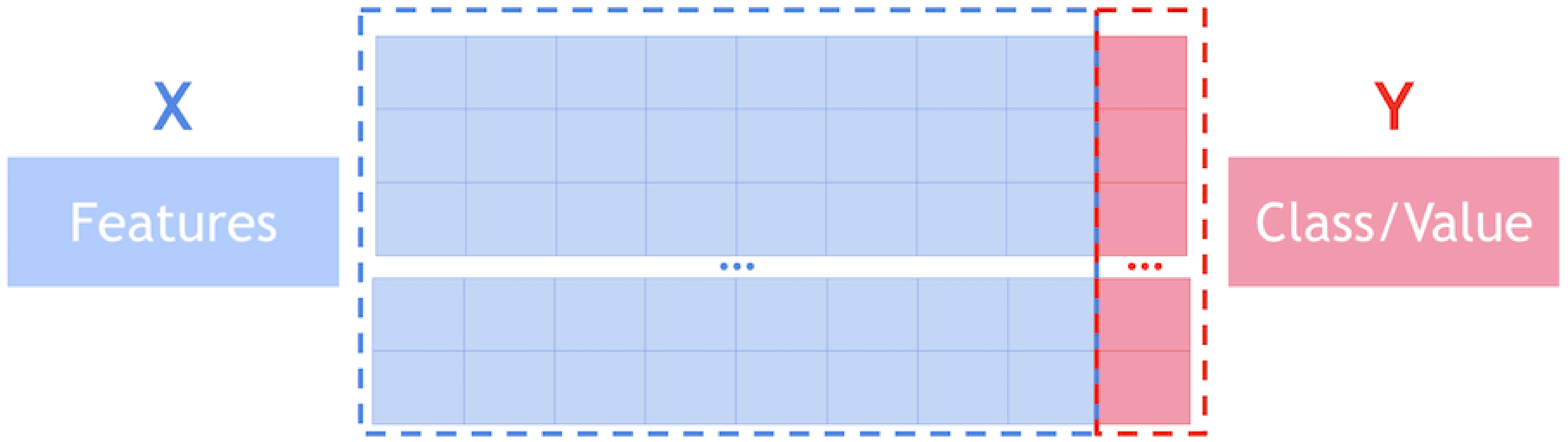
## Labeled Dataset



## A Bit of Notation

- We indicate by $X \in \mathbb{R}_{m,n}$ the $m$-by-$n$ **feature matrix** resulting from the $m$ $n$-dimensional feature vectors.
- Similarly, we denote by $Y$ the $m$-dimensional vector containing instance labels.

## Labeled Dataset



## 3. Model Training

## Intuition

- **Idea**: There exists an **unknown target function** $f$, which maps element of $X$ to elements of $Y$.

$$f = X \mapsto Y$$

- Unfortunately, we cannot just write down an algorithm that implements $f$.
- Still, we can try to ***learn*** $f$, namely to find another function $h^*$ which approximates the *true* $f$ ($h^* \approx f$) using the data $\mathcal{D}$ we observed.
- We refer to $h^*$ as our **hypothesis** (used to approximate $f$).

## How Do We Select $h^*$ From?

- We choose $h^*$ from a family of functions $\mathcal{H}$, called the **hypothesis space**.
- To actually pick $h^*$, we need to further specifying two components:
  - the **loss function** (a.k.a. **cost function**) denoted by $\ell$
  - the **learning algorithm** denoted by $\mathcal{A}$

## The Loss Function $\ell$

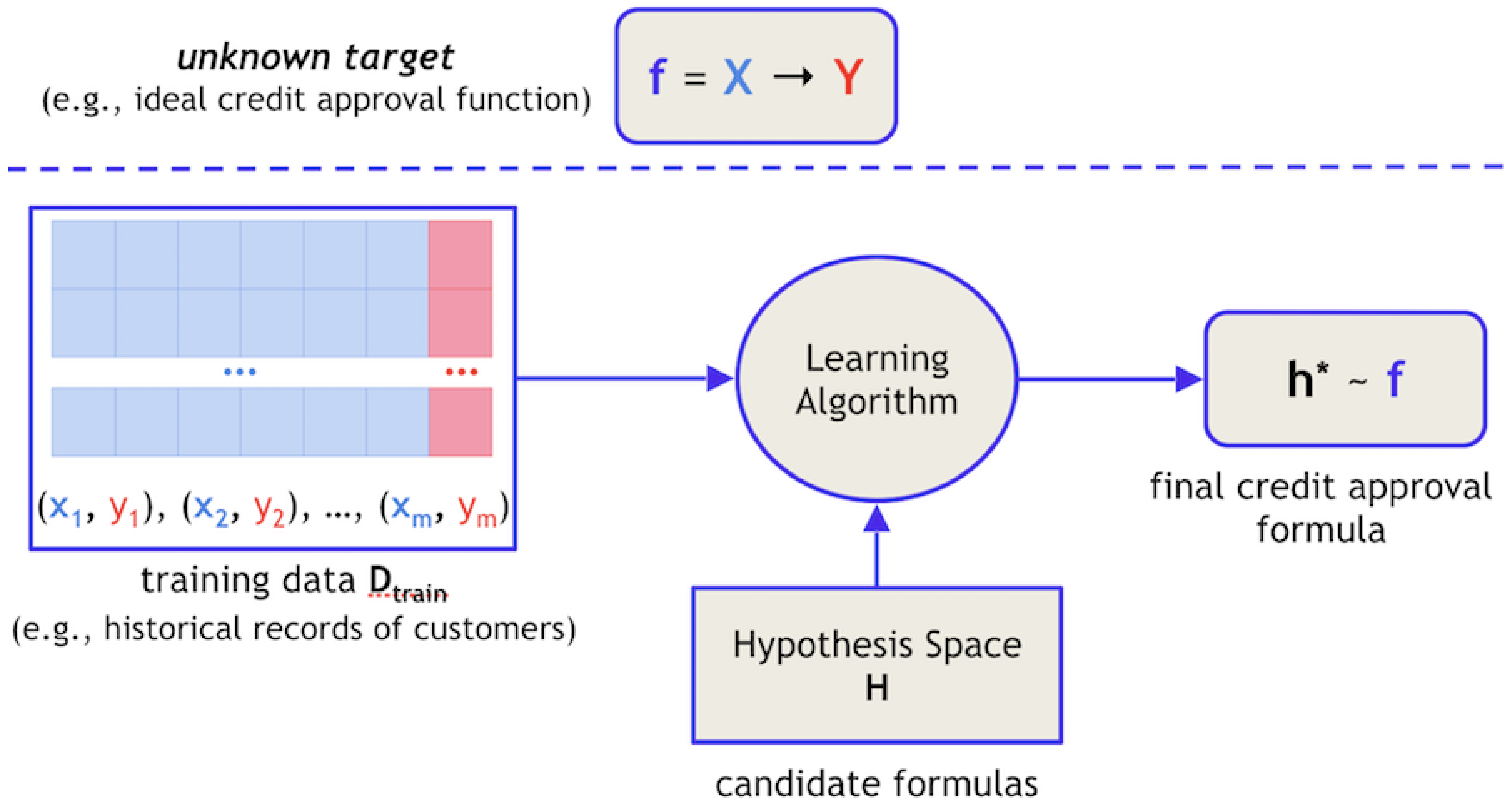
- Measures the error we would make if a hypothesis $h$ is used instead of the true $f$.
- Such an error can be computed *only* on the data we observed (i.e., on $\mathcal{D}$).
- Therefore, it depends on the hypothesis *and* the dataset, i.e., $\ell : \mathcal{H} \times \mathcal{D} \mapsto \mathbb{R}$.
- We should use this **in-sample error** (a.k.a. **empirical loss**) as an ***estimate*** of the **out-of-sample error** (a.k.a. **expected loss** or **risk**).

## The Learning Algorithm $\mathcal{A}$

- Defines the strategy we use to search the hypothesis space $\mathcal{H}$ for picking our **best** hypothesis $h^* \in \mathcal{H}$.
- Here, "**best**" means the hypothesis that **minimizes** the loss function on the observed data (**Empirical Risk Minimization**).
- In other words, among all the hypotheses specified by $\mathcal{H}$, the learning algorithm will pick the one that minimizes $\ell$.

$$h^* = \mathrm{argmin}_{h \in \mathcal{H}} \ell(h, \mathcal{D})$$

# One-Slide ML



*unknown target*
(e.g., ideal credit approval function)
$f = X \to Y$
$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$
training data $D_{train}$
(e.g., historical records of customers)
Learning Algorithm
$h^* \sim f$
final credit approval formula
Hypothesis Space H
candidate formulas

## The Hypothesis Space $\mathcal{H}$

- Intuitively, the larger the hypothesis space:
    - the larger will be the set of functions that can be represented (**+**)
    - the harder will be for the learning algorithm to pick the best $h^*$ (**-**)
- **Trade-off**: Put some constraints on $\mathcal{H}$, e.g., limit the search space $\mathcal{A}$ has to explore only to **linear functions**.

## Example of $\mathcal{H}$: Linear Models

- The underlying assumption here is that there exists a **linear relationship** between $X$ (**features**) and $Y$ (**class label**/**value**).

$$\mathcal{H} = \{h_{\boldsymbol{\theta}} : X \longmapsto Y \mid h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n\}$$

- $\boldsymbol{\theta}$ is called the **vector of parameters** (of the linear model).
- $x_0 = 1$ by convention.
- Among all the possible instantiations of $\boldsymbol{\theta}$, the learning algorithm will pick $\boldsymbol{\theta}^*$ as the one which minimizes the **loss function** $\ell$ computed on $\mathcal{D}$.

## A Possible Loss Function: *Mean Squared Error* (MSE)

- **MSE** is just a possible (mathematically convenient) choice of loss function $\ell$.
- **MSE** measures the average error when the true target $f$ is replaced with a hypothesis $h_{\boldsymbol{\theta}} \in \mathcal{H}$ on the observed data $\mathcal{D}$.

$$\ell(h_{\boldsymbol{\theta}}, \mathcal{D}) = \text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D})$$

$$= \frac{1}{m} \sum_{i=1}^{m} \Big( h_{\boldsymbol{\theta}}(\mathbf{x}_i) - f(\mathbf{x}_i) \Big)^2$$

$$= \frac{1}{m} \sum_{i=1}^{m} \Big( h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i \Big)^2$$

- Each term of the above summation, i.e., $\Big( h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i \Big)^2$, represents the **squared error** w.r.t. a single instance in $\mathcal{D}$.

## The Learning Algorithm (Revisited)

- Reduces the learning problem to an **optimization problem**.
- Among all the possible $h_{\boldsymbol{\theta}} \in \mathcal{H}$, it picks $h^* = h_{\boldsymbol{\theta}^*}$, so as to minimize the loss function.

$$h^* = \texttt{argmin}_{\boldsymbol{\theta}}\{\ell(h_{\boldsymbol{\theta}}, \mathcal{D})\}$$

- In particular, if $\ell = \text{MSE}$:

$$h^* = \texttt{argmin}_{\boldsymbol{\theta}}\{\text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D})\} = \texttt{argmin}_{\boldsymbol{\theta}}\left\{\frac{1}{m}\sum_{i=1}^{m}\Big(h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i\Big)^2\right\}$$

## Minimum and Maximum of a Function: First Derivative

- Whenever we want to find the minimum (maximum) of a real-valued function $f : \mathbb{R} \longmapsto \mathbb{R}$, we look for **stationary points**, i.e., the set of points $S = \{(x, f(x)) \mid f'(x) = 0\}$ where the first derivative is 0.
- Note that depending on the shape of the function $f$, a stationary point doesn't necessarily correspond to a local minimum (maximum) but it may be a **point of inflection**.
- To investigate better what happens at a specific stationary point, we need to check the second derivative $f''(x)$.

## Minimum and Maximum of a Function: Second Derivative

- Given a stationary point $(x_s, f(x_s)) \in S$:
  - if $f''(x_s) > 0$ then $(x_s, f(x_s))$ is a **local minimum**
  - if $f''(x_s) < 0$ then $(x_s, f(x_s))$ is a **local maximum**
  - if $f''(x_s) = 0$ then $(x_s, f(x_s))$ may be an **inflection point**

## Minimum and Maximum of a Convex/Concave Function

- Any local minimum (maximum) of a **convex** (**concave**) function is also a **global** minimum (maximum).
- If we know the function is **convex** (**concave**) finding the minimum (maximum) can be done just by computing the first derivative and set it to 0.
- In the case of a multivariate function, this generalizes to compute the **gradient** ($\nabla$) of the function and set it to 0.

## The Gradient $\nabla$

- The gradient of a multivariate function $f : \mathbb{R}^n \longmapsto \mathbb{R}$ is the $n$-dimensional vector of the **partial derivatives** of the function w.r.t. each of its variable.

$$\nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

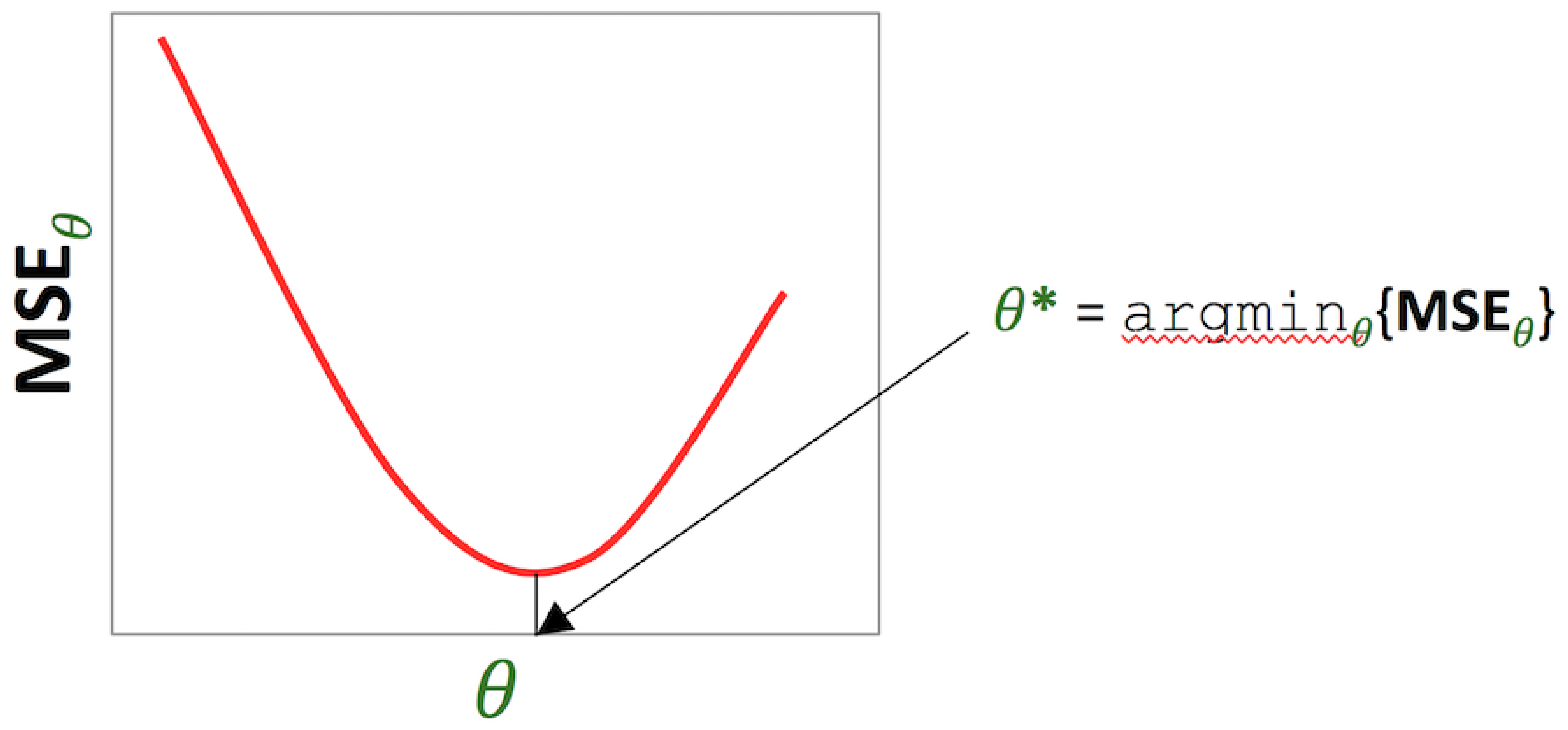
- Solving $\nabla f = \mathbf{0}$ means finding an $n$-dimensional vector resulting from the following:

$$\nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right) = \underbrace{(0, 0, \ldots, 0)}_{n} = \mathbf{0}$$

## $\mathrm{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D})$ is a Convex Function

- Each term of the summation of $\mathrm{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D})$ is a multivariate linear function of the model parameters $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_n)$.
- Linear functions are **convex**, and so does the square of a convex function; also, the sum of convex functions is again convex.
- Convex functions have a **unique local minimum**, which therefore happens to be the **global minimum**.

## Visualize The Optimization Problem



## Computing the Gradient of MSE

- The following is the expression of the gradient of $\text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D})$:

$$\nabla \text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D}) = \nabla \left[ \frac{1}{m} \sum_{i=1}^{m} \Big( h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i \Big)^2 \right]$$

## Computing the Gradient of MSE

$$\nabla \text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D}) = \nabla \left[ \frac{1}{m} \sum_{i=1}^{m} \Big( h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i \Big)^2 \right]$$

- Due to the **scalar multiple rule** and **sum rule** of the derivative, i.e., $\frac{\partial f}{\partial t}(\alpha t) = \alpha \frac{\partial f}{\partial t}$ ($\alpha \in \mathbb{R}$, constant) and $\frac{\partial f}{\partial t}\Big(\sum t\Big) = \sum \Big(\frac{\partial f}{\partial t}\Big)$, we can rewrite the above as follows:

$$\nabla \text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D}) = \frac{1}{m} \left[ \sum_{i=1}^{m} \nabla \Big( h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i \Big)^2 \right]$$

## Computing the Gradient of MSE (single instance)

- To make things easier, let's consider for a moment the special case where $\mathcal{D}$ contains just a **single** instsance, i.e., $\mathcal{D} = \{(\mathbf{x}, y)\}$. Therefore:

$$\nabla\text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D}) = \nabla\Big(h_{\boldsymbol{\theta}}(\mathbf{x}) - y\Big)^2$$

- Due to the **power rule** and **chain rule** of the derivative, i.e., $\frac{\partial f(t^\alpha)}{\partial t} = \alpha t^{\alpha-1}$ $\frac{\partial f(g(t))}{\partial t} = \frac{\partial f(g(t))}{\partial g(t)} \cdot \frac{\partial g(t)}{\partial t}$, we can rewrite the above as follows:

$$= 2\Big(h_{\boldsymbol{\theta}}(\mathbf{x}) - y\Big)\nabla\Big(h_{\boldsymbol{\theta}}(\mathbf{x}) - y\Big)$$

## Computing the Gradient of MSE (single instance)

$$\nabla\Big(h_{\boldsymbol{\theta}}(\mathbf{x}) - y\Big) = \nabla\Big(\theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n - y\Big)$$

$$= \Big( \frac{\partial(\theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n - y)}{\partial \theta_0}, \frac{\partial(\theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n - y)}{\partial \theta_1}, \ldots$$

$$\frac{\partial(\theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n - y)}{\partial \theta_n} \Big) = \Big(x_0, x_1, \ldots, x_n\Big) = \mathbf{x}$$

- Overall:

$$\nabla \text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D}) = 2\underbrace{\Big(h_{\boldsymbol{\theta}}(\mathbf{x}) - y\Big)}_{\text{scalar}} \cdot \underbrace{\mathbf{x}}_{(n+1)\text{-dimensional vector}}$$

## Computing the Gradient of MSE (single instance)

- We can rewrite the formula above using **vectorized notation**, noticing that $h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 x_0 + \theta_1 x_1 + \ldots \theta_n x_n = \boldsymbol{\theta}^T \mathbf{x}$.
- $\boldsymbol{\theta}^T \mathbf{x}$ is obviously a scalar which results from the dot product of a $1$-by-$(n+1)$ row vector $\boldsymbol{\theta}^T$ and an $(n+1)$-by-$1$ column vector $\mathbf{x}$.

$$\nabla \text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D}) = 2(\boldsymbol{\theta}^T \mathbf{x} - y) \cdot \mathbf{x}$$

## Computing the Gradient of MSE (single instance)

- The product of a vector by a scalar is applied element-wise, therefore:

$$\nabla \text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D}) = \begin{bmatrix} 2(\boldsymbol{\theta}^T \mathbf{x} - y) \cdot x_0 \\ 2(\boldsymbol{\theta}^T \mathbf{x} - y) \cdot x_1 \\ \vdots \\ 2(\boldsymbol{\theta}^T \mathbf{x} - y) \cdot x_n \end{bmatrix} = \begin{bmatrix} 2(\boldsymbol{\theta}^T \mathbf{x} - y) \\ 2(\boldsymbol{\theta}^T \mathbf{x} - y) \cdot x_1 \\ \vdots \\ 2(\boldsymbol{\theta}^T \mathbf{x} - y) \cdot x_n \end{bmatrix}$$

as we set $x_0 = 1$.

- Note that $\nabla \text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D})$ is an $(n+1)$-dimensional vector, as expected.

## Setting the Gradient of MSE to 0 (single instance)

- To find the local minimum (which is also global) of $\text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D})$, we set:

$$\nabla \text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D}) = \begin{bmatrix} 2(\boldsymbol{\theta}^T \mathbf{x} - y) \\ 2(\boldsymbol{\theta}^T \mathbf{x} - y) \cdot x_1 \\ \vdots \\ 2(\boldsymbol{\theta}^T \mathbf{x} - y) \cdot x_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \mathbf{0}$$

- Therefore, we have to solve a system of $n+1$ linear equations of $n+1$ variables, each equation of the form $2(\boldsymbol{\theta}^T \mathbf{x} - y) \cdot x_j = 0$, $j \in \{0, 1, \ldots, n\}$

## Computing the Gradient of MSE ($m$ instances)

- Going back to the overall **MSE** in the general case where $\mathcal{D}$ contains $m$ instances:

$$\nabla \text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D}) = \frac{1}{m}\Bigg\{ \Bigg[2\Big(h_{\boldsymbol{\theta}}(\mathbf{x}_1) - y_1\Big)\nabla\Big(h_{\boldsymbol{\theta}}(\mathbf{x}_1) - y_1\Big)\Bigg] +$$

$$\ldots$$

$$+\Bigg[2\Big(h_{\boldsymbol{\theta}}(\mathbf{x}_m) - y_m\Big)\nabla\Big(h_{\boldsymbol{\theta}}(\mathbf{x}_m) - y_m\Big)\Bigg]\Bigg\}$$

$$= \frac{2}{m}\Bigg[\sum_{i=1}^{m}\Big(h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i\Big)\nabla\Big(h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i\Big)\Bigg]$$

## Computing the Gradient of MSE ($m$ instances)

- Putting all together:

$$\nabla \text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D}) = \frac{2}{m}\left[\sum_{i=1}^{m} \underbrace{\Big(h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i\Big)}_{\text{scalar}} \cdot \underbrace{\mathbf{x}_i}_{(n+1)\text{-dimensional vector}}\right]$$

$$\nabla \text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D}) = \begin{bmatrix} \frac{2}{m}(\boldsymbol{\theta}^T\mathbf{x_1} - y_1)\cdot x_{1,0} + \ldots + \frac{2}{m}(\boldsymbol{\theta}^T\mathbf{x_m} - y_m)\cdot x_{m,0} \\ \frac{2}{m}(\boldsymbol{\theta}^T\mathbf{x_1} - y_1)\cdot x_{1,1} + \ldots + \frac{2}{m}(\boldsymbol{\theta}^T\mathbf{x_m} - y_m)\cdot x_{m,1} \\ \vdots \\ \frac{2}{m}(\boldsymbol{\theta}^T\mathbf{x_1} - y_1)\cdot x_{1,n} + \ldots + \frac{2}{m}(\boldsymbol{\theta}^T\mathbf{x_m} - y_m)\cdot x_{m,n} \end{bmatrix}$$

## Computing the Gradient of MSE ($m$ instances)

- Note that we have introduced a second subscript to be able to index each feature's instance within $\mathcal{D}$, i.e., $x_{i,j}$.
- Moreover, we have set all $x_{i,0} = 1 \; (i \in \{1, \ldots, m\})$.

$$\nabla \text{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D}) = \frac{2}{m} \begin{bmatrix} (\boldsymbol{\theta}^T \mathbf{x_1} - y_1) + \ldots + (\boldsymbol{\theta}^T \mathbf{x_m} - y_m) \\ (\boldsymbol{\theta}^T \mathbf{x_1} - y_1) \cdot x_{1,1} + \ldots + (\boldsymbol{\theta}^T \mathbf{x_m} - y_m) \cdot x_{m,1} \\ \vdots \\ (\boldsymbol{\theta}^T \mathbf{x_1} - y_1) \cdot x_{1,n} + \ldots + (\boldsymbol{\theta}^T \mathbf{x_m} - y_m) \cdot x_{m,n} \end{bmatrix}$$

## Setting the Gradient of MSE to 0 ($m$ instances)

$$\underbrace{\frac{2}{m}\begin{bmatrix} (\boldsymbol{\theta}^T\mathbf{x_1} - y_1) + \ldots + (\boldsymbol{\theta}^T\mathbf{x_m} - y_m) \\ (\boldsymbol{\theta}^T\mathbf{x_1} - y_1)\cdot x_{1,1} + \ldots + (\boldsymbol{\theta}^T\mathbf{x_m} - y_m)\cdot x_{m,1} \\ \vdots \\ (\boldsymbol{\theta}^T\mathbf{x_1} - y_1)\cdot x_{1,n} + \ldots + (\boldsymbol{\theta}^T\mathbf{x_m} - y_m)\cdot x_{m,n} \end{bmatrix}}_{\nabla \text{MSE}(h_{\boldsymbol{\theta}},\mathcal{D})} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \mathbf{0}$$

- Again, we have to solve a system of $n+1$ linear equations of $n+1$ variables, each equation of the form $\frac{2}{m}\left[(\boldsymbol{\theta}^T\mathbf{x_1} - y_1)\cdot x_{1,j} + \ldots + (\boldsymbol{\theta}^T\mathbf{x_m} - y_m)\cdot x_{m,j}\right] = 0,$ $j \in \{0, 1, \ldots, n\}$

## Vectorized Form of the Gradient of MSE

$$\underbrace{\begin{bmatrix} x_{1,0}, x_{1,1}, \ldots, x_{1,n} \\ x_{2,0}, x_{2,1}, \ldots, x_{2,n} \\ \vdots \\ x_{m,0}, x_{m,1}, \ldots, x_{m,n} \end{bmatrix}}_{\textbf{feature matrix } \mathbf{X}_{m,n+1}} \quad \underbrace{\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}}_{\textbf{parameter vector } \boldsymbol{\theta}} \quad \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}}_{\textbf{label vector } \mathbf{y}_{m,1}}$$

## Vectorized Form of the Gradient of MSE

$$\nabla \mathsf{MSE}(h_{\boldsymbol{\theta}}, \mathcal{D}) = \frac{2}{m}\mathbf{X}^T(\mathbf{X} \cdot \boldsymbol{\theta} - \mathbf{y})$$

$$\frac{2}{m}\mathbf{X}^T(\mathbf{X} \cdot \boldsymbol{\theta} - \mathbf{y}) = \mathbf{0}$$

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

$$\boldsymbol{\theta} = \mathbf{X}^\dagger \cdot \mathbf{y},\ \text{where } \mathbf{X}^\dagger = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \text{ is the } \textbf{pseudo-inverse} \text{ of } \mathbf{X}$$

## Additional Notes

- Luckily, there exists a closed-form solution to the convex optimization problem above (i.e., to minimize **MSE**).
- However, other choices of loss functions (even if convex) may need an **iterative** approach to get to a (local) minimum.
- For example, **gradient descent** use the gradient to **iteratively** converge to a local minimum (guaranteed for convex loss functions).
- We should use gradient descent as the size of our problem increases because computing the inverse of a large matrix is generally a very costly task ($O(n^3)$, where $n \times n$ is the size of the matrix).

## Gradient Descent

- A generic iterative algorithm used to find an approximate solution to an optimization problem.
- It consists of 3 parameters:
    - $\boldsymbol{\theta}_0 \in \mathbb{R}^n$ as the **initial (random) guess**
    - $\epsilon > 0$ **stopping criterion**
    - $\alpha$ **step size**

## Gradient Descent: The Algorithm

```
1. $\boldsymbol{\theta}^* = \boldsymbol{\theta}_0$;
2. while $||\nabla f||_2 > \epsilon$: // replace $f$ with the function of interest, e.g., MSE$(h_{\boldsymbol{\theta}}, \mathcal{D})$
       $\boldsymbol{\theta}^* = \boldsymbol{\theta}^* - \underbrace{\alpha}_{\text{step size}} \nabla f(\boldsymbol{\theta}^*)$; // the direction of greatest decrease of $f$ is opposite to the gradient vector
3. return $\boldsymbol{\theta}^*$;
```

## Gradient Descent: The Choice of $\boldsymbol{\theta}_0$

- The starting point $\boldsymbol{\theta}_0$ can be chosen arbitrarily, though for non-convex function $f$ the output of gradient descent can vary with its choice.
- For convex $f$, gradient descent will converge toward the **same point**, i.e., the global minimum, independently of the starting point.
- The choice of starting point can still affect the number of iterations until convergence.
- In practice, one should choose $\boldsymbol{\theta}_0$ according to our best guess as to where the global minimum is likely to be.

## Gradient Descent: The Choice of $\epsilon$

- The parameter $\epsilon$ determines the stopping criterion.
- Note that since $\epsilon > 0$, gradient descent generally does not halt at an **actual** local minimum, but rather at some kind of "approximate local minimum".
- Smaller values of $\epsilon$ mean more iterations before stopping but a higher-quality solution at termination.
- In practice, one tries various values of $\epsilon$ to achieve the right balance between computation time and solution quality.

## Gradient Descent: The Choice of $\alpha$

- The final parameter $\alpha$, the "step size", is perhaps the most important.
- While gradient descent is flexible enough that different values of $\alpha$ can be used in different iterations, in practice one typically sets $\alpha$ in advance and stick to it over all iterations.
- The "best" value of $\alpha$ is typically chosen by experimentation. For example, we can run the entire gradient descent algorithm with a few different choices of $\alpha$ to see which run gives us the best results.

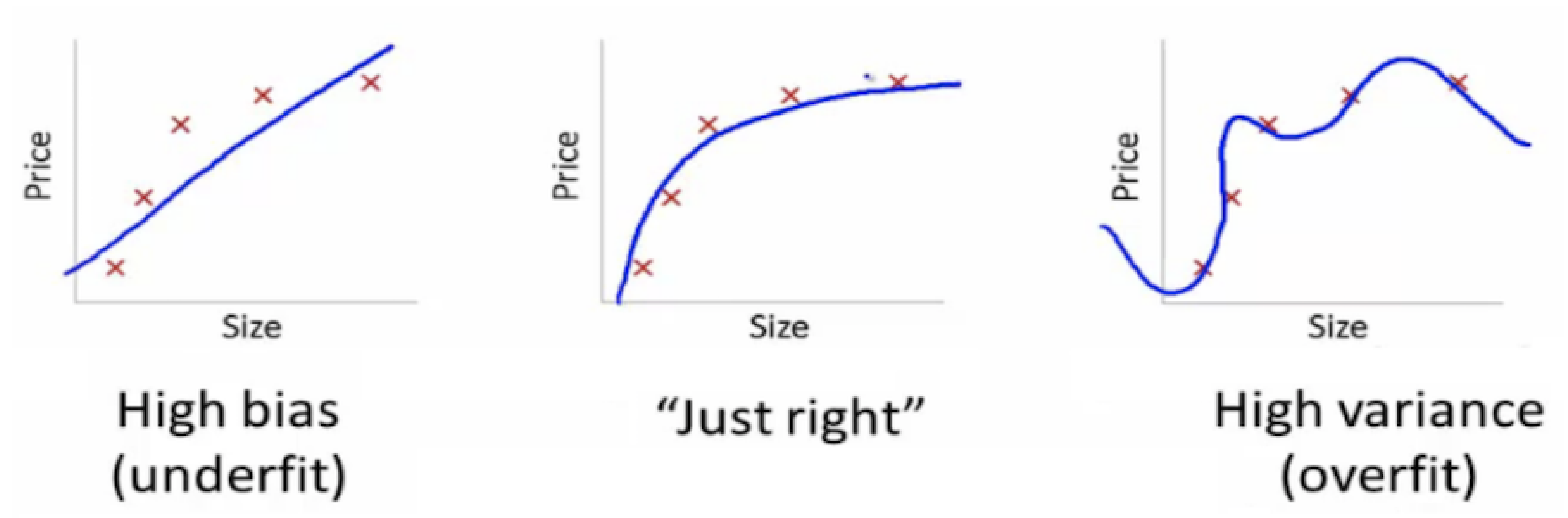
## Overfitting (High Variance)

- Minimizing the loss function considering the whole dataset $\mathcal{D}$ just limits the **in-sample error**.
- Our ultimate goal is to pick a hypothesis $h^*$ which is able to **generalize** to unseen instances (i.e., minimize the **out-of-sample error**).
- Note that if we pick a hypothesis that just memorizes all the instances in $\mathcal{D}$, this will have a 0 **in-sample error** but this is **not** learning!
- **Overfitting**: $h^*$ is not learning the true $f$ but just mimic random noise
- **High Variance**: $h^*$ does not generalize, as it strongly depends on $\mathcal{D}$

## Underfitting (High Bias)

- Anyway, we don't want $h^*$ to perform poorly on $\mathcal{D}$.
- **Underfitting**: $h^*$ has a very high in-sample error
- **High Bias**: $h^*$ does not take advantage of $\mathcal{D}$

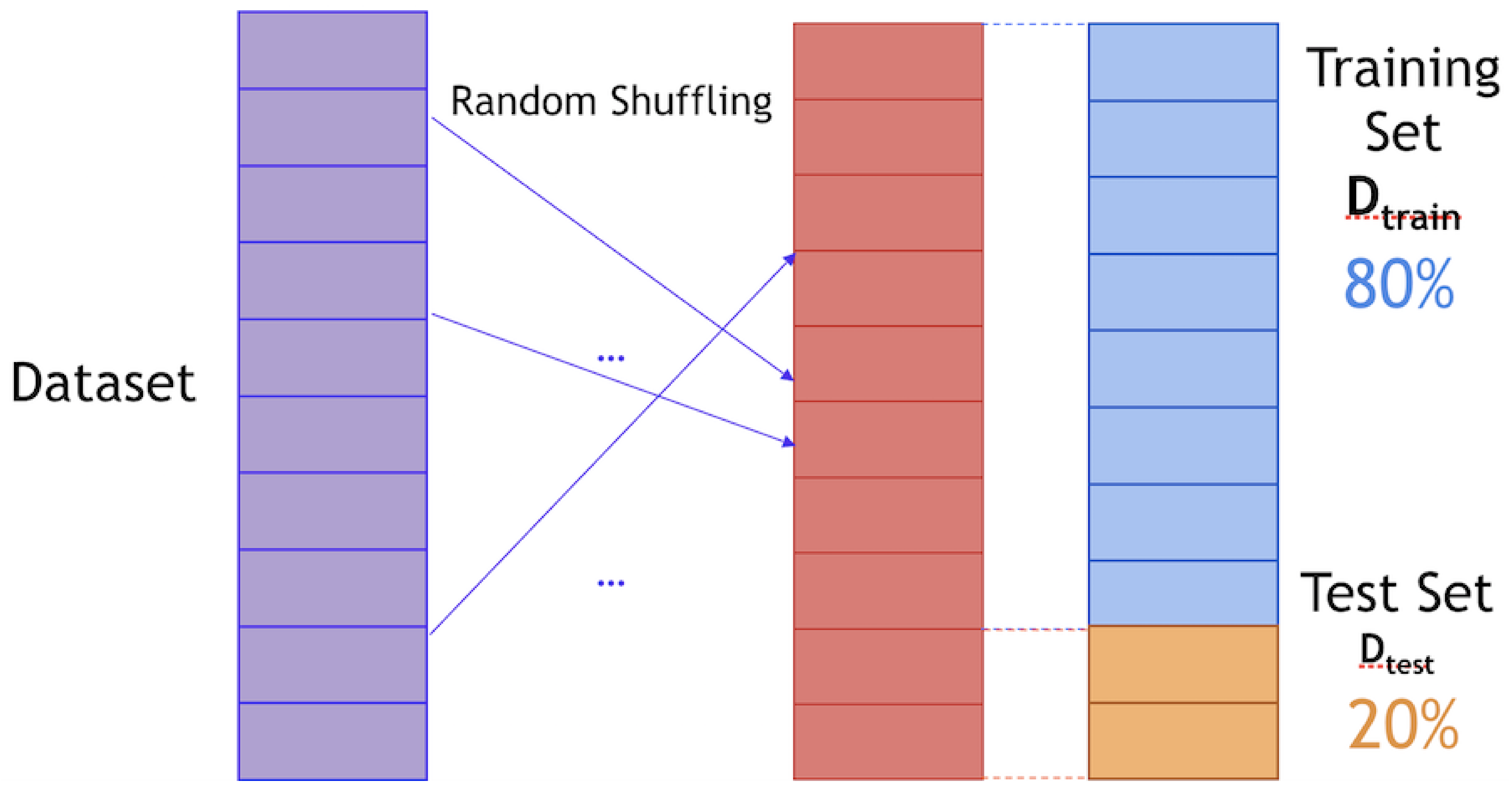
## Bias-Variance Tradeoff



## Splitting the Dataset $\mathcal{D}$

- To prevent overfitting, split the dataset into (at least) two separate random samples:
  - **Training Set** ($\mathcal{D}_{\text{train}}$) used to learn $h^*$, such that $|\mathcal{D}_{\text{train}}|$ = 0.8 $|\mathcal{D}|$
  - **Test** or **Held-out Set** ($\mathcal{D}_{\text{test}}$) used to evaluate $h^*$, such that $|\mathcal{D}_{\text{test}}|$ = $|\mathcal{D}| - |\mathcal{D}_{\text{train}}|$
- Note that the (*offline* or *online*) metrics we use to evaluate $h^*$ do not necessarily have to be the same of that we optimize at training time.

## Splitting the Dataset $D$



## How Much Data Do We Need?

- In general, the more data we have the better we learn.
- Even if the datasets we operate on are not so big, we can still take advantage of ML.
- For example, we can use a technique called $k$-**fold cross validation** (e.g., $k = 10$) instead of just splitting $D$ in two datasets.

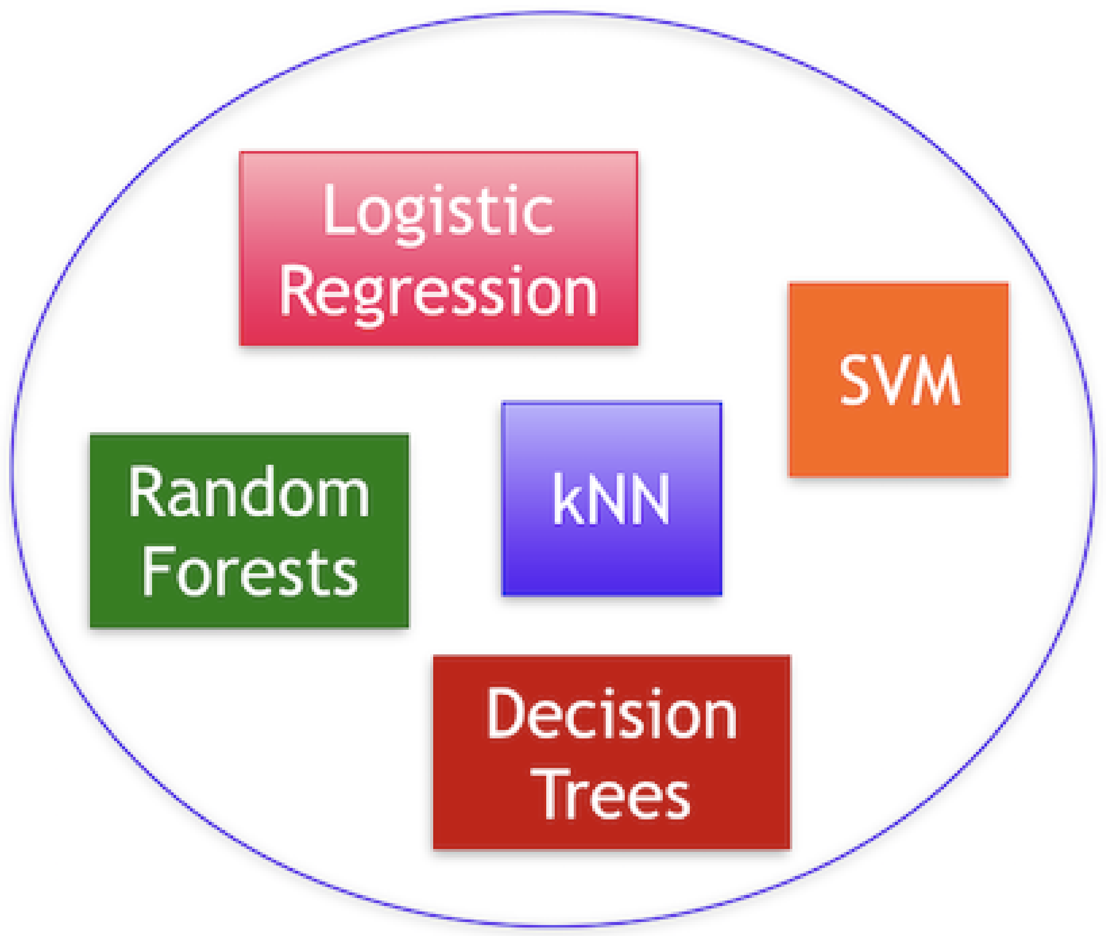
## $k$-fold Cross Validation

- (Optional) Split the dataset in $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$, as discussed above.
- Repeat for $k$ iterations the following step:
    - Randomly select a proportion of $1/k$ instances from $\mathcal{D}_{\text{train}}$ which will be used for testing, whilst the remaining $(k-1)/k$ proportion of the dataset is used for training.
    - Compute the error on the test proportion.
- Eventually, compute the **cross validation** error as the average of the individual errors obtained in each of the $k$ runs.

## 4. Model Selection/Evaluation

## Which Model Should I Use?

- Several learning models are designed to achieve the same tasks.
- Each learning model has its own set of parameters, a.k.a. **hyperparameters** (e.g., the number $k$ of neighbors in $k$NN).



## Validation Set

- It is crucial not to mix model training with hyperparameter sweeping.
- Another portion of the original dataset (**Validation Set**) should be reserved for this task.
- This can be also done within each step of a $k$-fold cross validation.

## Example

Suppose we want to select which value of $k \in \{2, 5, 10\}$ of a $k$NN gives the best performance.

1. Train a separate model for each value of $k$ on the training set (e.g., 70%)
2. Measure the error of each model on the validation set (e.g., 10%)
3. Select the model whose value of $k$ gives the best performance on the validation set (e.g., $k = 5$)
4. Re-train **only** this model on the training + validation sets
5. Measure the performance on the test set (e.g., 20%)

## Example Using Cross Validation

1. For each cross validation run:
    - Train a separate model for each value of $k$ on the training set portion of that run
    - Measure the error of each model on the validation set portion of that run
2. Select the model whose value of $k$ gives the best performance on the 10 averaged validation sets (e.g., $k = 5$)
3. Re-train **only** this model on the whole training set (e.g., 80%)
4. Measure the performance on the test set (e.g., 20%)