Introduction to Machine Learning

Jindřich Libovický (reusing materials by **Milan Straka**)

📅 **October 03, 2023**

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Course Objectives: What you will learn

After this course you should...

- Be able to reason about task/problems **suitable for ML**
  - Know when to use classification, regression and clustering
  - Be able to choose from this method Linear and Logistic Regression, Multilayer Perceptron, Nearest Neighbors, Naive Bayes, Gradient Boosted Decision Trees, k-means clustering
- Think about learning as (mostly probabilistic) **optimization on training data**
  - Know how the ML methods learn including theoretical explanation
- Know how to properly **evaluate** ML
  - Think about generalization (and avoiding overfitting)
  - Be able to choose a suitable evaluation metric
  - Responsibly decide what model is better
- Be able to **implement ML algorithms** on a conceptual level
- Be able to **use Scikit-learn** to solve ML problems in Python

Course Objectives: What you will *not* learn

- Data Science – How to (ethically, legally, effeciently, etc.) get data and how to **clean** data.
- More advanced neural networks (and how ChatGPT works) – this is covered in NPFL114
- How to apply ML in your specific field / your business

Organization

Course Website: https://ufal.mff.cuni.cz/courses/npfl129

- Slides, recordings, assignments, exam questions

Course Repository: https://github.com/ufal/npfl129

- Templates for the assignments, slide sources.

Piazza

- Piazza will be used as a communication platform.

  You can post questions or notes,
  - **privately** to the instructors,
  - **publicly** to everyone (signed or anonymously).
    - Other students can answer these too, which allows you to get faster response.
    - However, **do not include even parts of your source code** in public questions.
- Please use Piazza for **all communication** with the instructors.
- You will get the invite link after the first lecture.

ReCodEx

https://recodex.mff.cuni.cz

- The assignments will be evaluated automatically in ReCodEx.
- If you have a MFF SIS account, you should be able to create an account using your CAS credentials and should automatically see the right group.
- Otherwise, there will be **instructions** on **Piazza** how to get ReCodEx account (generally you will need to send me a message with several pieces of information and I will send it to ReCodEx administrators in batches).

Course Requirements

Practicals

- There will be about 2-3 assignments a week, each with a 2-week deadline.
    - There is also another week-long second deadline, but for fewer points.
- After solving the assignment, you get non-bonus points, and sometimes also bonus points.
- To pass the **practicals, you need to get 70 non-bonus points**. There will be assignments for at least 105 non-bonus points.
- If you get **more than 70 points** (be it bonus or non-bonus), they will be *transferred to the exam* (but at most 40 points are transferred).

Lecture

You need to pass a written exam.

- All questions are publicly listed on the course website.
- There are questions for 100 points in every exam, plus at most 40 surplus points from the practicals and plus at most 10 surplus points for **community work** (improving slides, ...).
- You need 60/75/90 points to pass with grade 3/2/1.

Today's Lecture Objectives

After this lecture you should be able to

- Explain to an non-expert what machine learning is
- Explain the difference between classification and regression
- Implement a simple linear-algebra-based algorithm for training linear regression

Machine Learning

A possible definition of learning from Mitchell (1997):

> 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.

- Task T
  - *classification*: assigning one of k categories to a given input
  - *regression*: producing a number $x \in \mathbb{R}$ for a given input
  - *structured prediction*, *denoising*, *density estimation*, …
- Measure P
  - *accuracy*, *error rate*, *F-score*, …
- Experience E
  - *supervised*: usually a dataset with desired outcomes (*labels* or *targets*)
  - *unsupervised*: usually data without any annotation (raw text, raw images, …)
  - *reinforcement learning*, *semi-supervised learning*, …

Intuitive definition: ML vs Standard Programming

Programming

- We can **formally describe** a problem with clear concepts
- Program = unambiguous set of **instructions** that handles the concepts

Example - e-shop: Concepts: goods, store, customer, order, ...
Simple algorithms: place an order, send an order, ...

Machine Learning

- We have **data** and a **measure** how our problem is solved
- Typically, we do not know how to code that solves the problem

Example - machine translation there is not set of formal instruction how to translate, there is a lot of data

Supervised Machine Learning

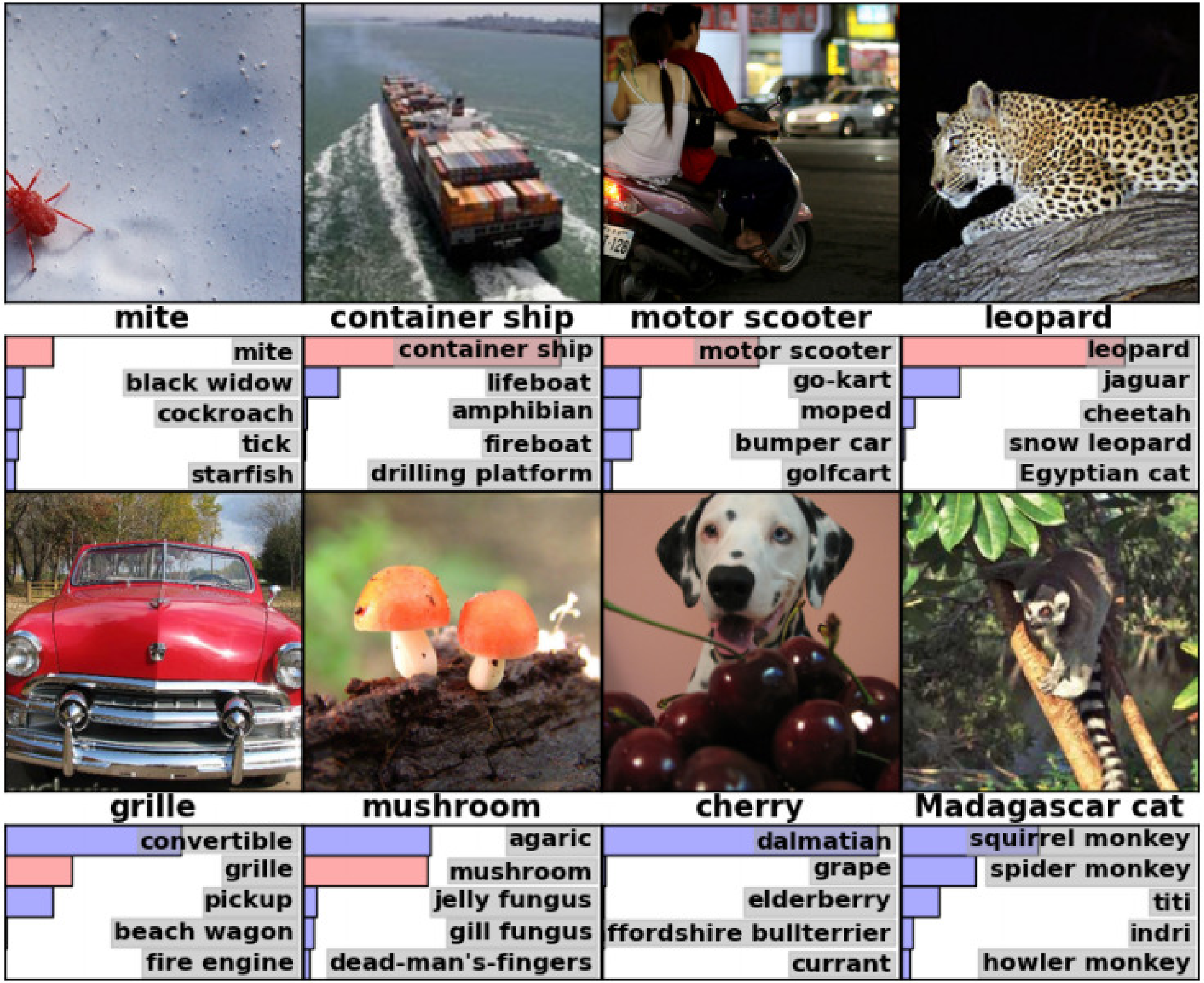
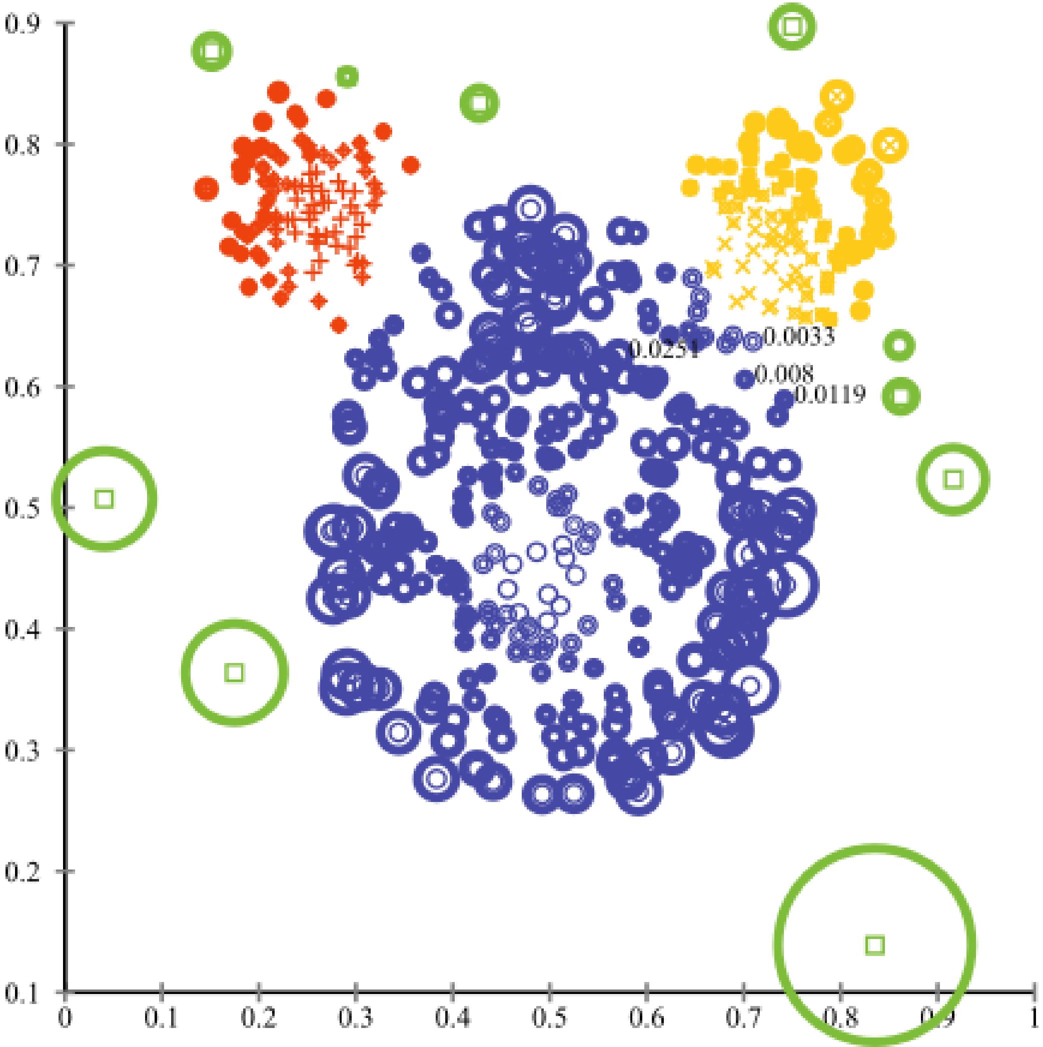


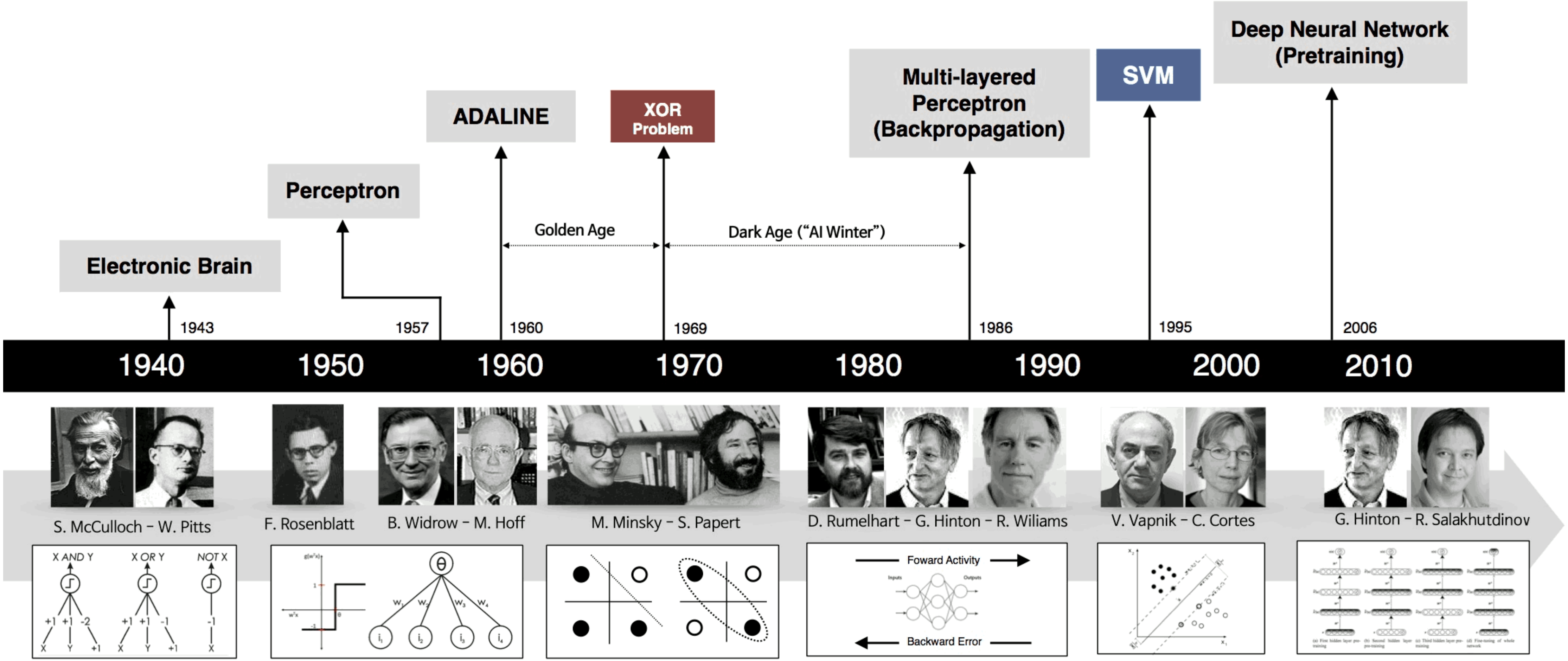
Figure 4 of "ImageNet Classification with Deep Convolutional Neural Networks" by Alex Krizhevsky et al.

Unsupervised Machine Learning



Outlier detection https://elki-project.github.io

Introduction to Machine Learning History



Electronic Brain

1943

Perceptron

1957

ADALINE

1960

XOR Problem

1969

Golden Age

Dark Age (“AI Winter”)

Multi-layered Perceptron (Backpropagation)

1986

SVM

1995

Deep Neural Network (Pretraining)

2006

S. McCulloch – W. Pitts

- Adjustable Weights
- Weights are not Learned

F. Rosenblatt

B. Widrow – M. Hoff

- Learnable Weights and Threshold

M. Minsky – S. Papert

- XOR Problem

D. Rumelhart – G. Hinton – R. Wiliams

- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting

V. Vapnik – C. Cortes

- Limitations of learning prior knowledge
- Kernel function: Human Intervention

G. Hinton – R. Salakhutdinov

- Hierarchical feature Learning

Modified from https://www.slideshare.net/deview/251-implementing-deep-learning-using-cu-dnn/4

Basic Machine Learning Settings

Assume input of $\boldsymbol{x} \in \mathbb{R}^D$. The two basic ML tasks are:

1. **Regression**: The goal is to predict a real-valued target variable $t \in \mathbb{R}$ for given $\boldsymbol{x}$.
2. **Classification**: Assuming a fixed set of K labels, the goal is to choose a corresponding label/class for given $\boldsymbol{x}$.
   - We can predict the class only.
   - We can predict the whole distribution of all classes probabilities.

We usually have a **training set**:

- Consists of examples of $(\boldsymbol{x}, t)$
- Probabilistic interpretation: Generated independently from a **data-generating distribution**

Optimization vs. Machine Learning

- **Optimization** = match the training set as well as possible
- ML's goals is **generalization** = match **previously unseen data** as well as possible

↓

?□?

We typically estimate it using a **test set** of examples independent of the training set.
(in probabilistic interpretation generated by the same data-generating distribution)

Notation

- a, $\boldsymbol{a}$, $\boldsymbol{A}$, A: scalar (integer or real), vector, matrix, tensor
  - all vectors are always **column** vectors
  - transposition changes a column vector into a row vector, so $\boldsymbol{a}^T$ is a row vector
  - we denote the **dot (scalar) product** of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ using $\boldsymbol{a}^T\boldsymbol{b}$
    - we understand it as matrix multiplication
  - the $\|\boldsymbol{a}\|_2$ or just $\|\boldsymbol{a}\|$ is the Euclidean (or L^2) norm
    - $\|\boldsymbol{a}\|_2 = \sqrt{\sum_i a_i^2}$
- a, $\mathbf{a}$, $\mathbf{A}$: scalar, vector, matrix random variable
- $\mathbb{A}$: set; $\mathbb{R}$ is the set of real numbers, $\mathbb{C}$ is the set of complex numbers
- $\frac{df}{dx}$: derivative of f with respect to x
- $\frac{\partial f}{\partial x}$: partial derivative of f with respect to x
- $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$: gradient of f with respect to $\boldsymbol{x}$, i.e., $\left(\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial x_n}\right)$

Example Dataset

Assume we have the following data, generated from an underlying curve by adding a small amount of noise.

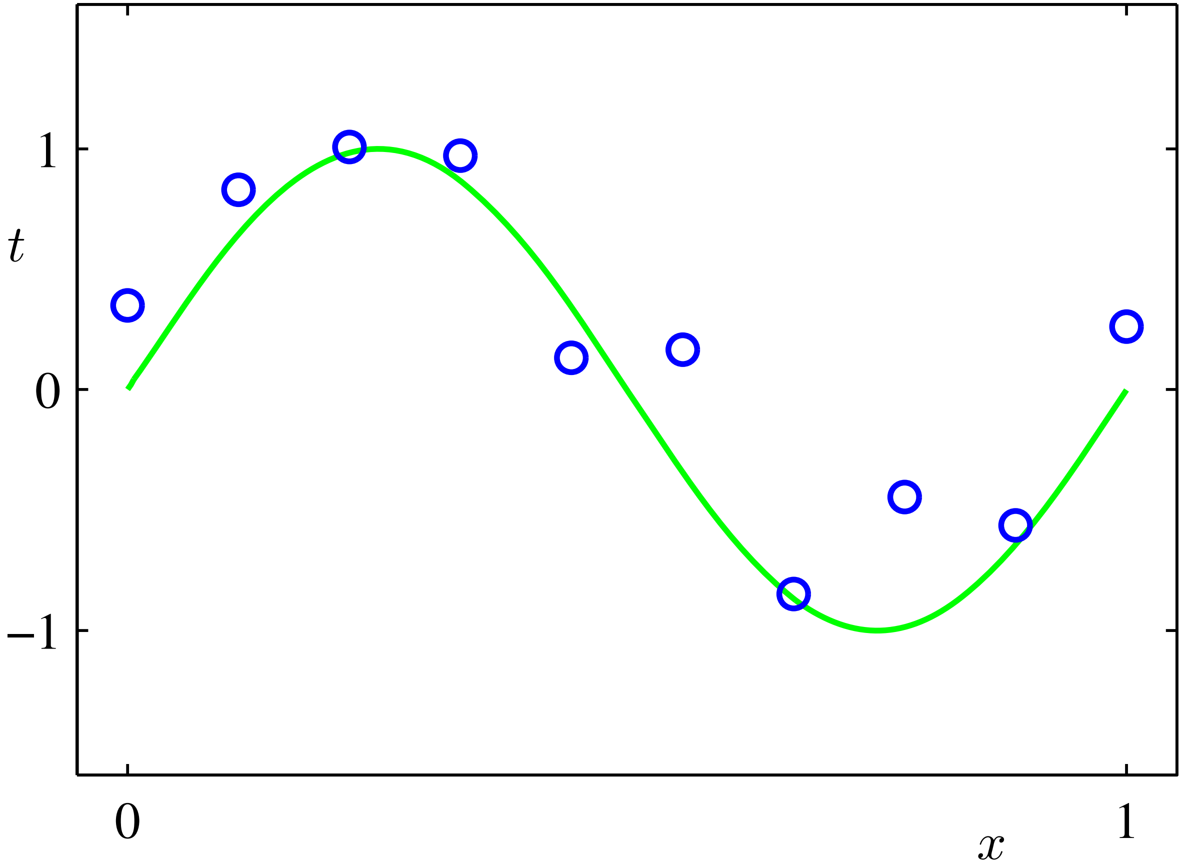


Figure 1.2 of Pattern Recognition and Machine Learning.

Input Data

Usually, ML algorithms are trained using the **train set** $\boldsymbol{X} \in \mathbb{R}^{N\times D}$: a collection of N instances, each represented by D real numbers.

In supervised learning, we also have a **target** $\boldsymbol{t}$ for every instance,

- a real number for regression, $\boldsymbol{t} \in \mathbb{R}^N$;
- a class for classification, $\boldsymbol{t} \in \{0, 1, \ldots, K-1\}^N$.

The input to ML learning algorithms is frequently preprocessed, i.e., the algorithms do not always work directly on the input $\boldsymbol{X}$, but on some modification of it. These are called **features**.

In some literature, processed inputs are called a **design matrix** $\boldsymbol{\Phi} \in \mathbb{R}^{N\times M}$, we will denote everything as $\boldsymbol{X}$.

Linear Regression

Given an input value $\boldsymbol{x} \in \mathbb{R}^D$, one of the simplest models to predict a target real value is **linear regression**:

$$y(\boldsymbol{x}; \boldsymbol{w}, b) = x_1 w_1 + x_2 w_2 + \ldots + x_D w_D + b = \sum_{i=1}^{D} x_i w_i + b = \boldsymbol{x}^T \boldsymbol{w} + b.$$

The $\boldsymbol{w}$ are usually called *weights* and b is called *bias*.

Sometimes it is convenient not to deal with the bias separately. Instead, we might enlarge the input vector $\boldsymbol{x}$ by padding a value 1, and consider only $\boldsymbol{x}^T \boldsymbol{w}$, the bias is encoded by the last weight. Therefore, "weights" often both weights and biases.

Separate Bias vs. Padding $\boldsymbol X$ with Ones

Using an explicit bias term in the form of $y(x) = \boldsymbol x^T \boldsymbol w + b$.

$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ & \vdots & \\ x_{n1} & x_{n2} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + b = \begin{bmatrix} w_1 x_{11} + w_2 x_{12} + b \\ w_1 x_{21} + w_2 x_{22} + b \\ \vdots \\ w_1 x_{n1} + w_2 x_{n2} + b \end{bmatrix}$$

With extra 1 padding in $\boldsymbol X$ and an additional b weight representing the bias.

$$\begin{bmatrix} x_{11} & x_{12} & 1 \\ x_{21} & x_{22} & 1 \\ & \vdots & \\ x_{n1} & x_{n2} & 1 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = \begin{bmatrix} w_1 x_{11} + w_2 x_{12} + b \\ w_1 x_{21} + w_2 x_{22} + b \\ \vdots \\ w_1 x_{n1} + w_2 x_{n2} + b \end{bmatrix}$$

Linear Regression

We have a dataset of N input values $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ and targets $t_1, \ldots, t_N$.

Find weight values = minimize an **error function** between the real target values and their predictions.

A popular and simple error function is *mean squared error*:

$$\mathrm{MSE}(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} \big(y(\boldsymbol{x}_i; \boldsymbol{w}) - t_i\big)^2.$$

Often, *sum of squares*

$$\frac{1}{2} \sum_{i=1}^{N} \big(y(\boldsymbol{x}_i; \boldsymbol{w}) - t_i\big)^2$$

is used instead. Minimizing it is equal to minimizing MSE, but the math comes out nicer.

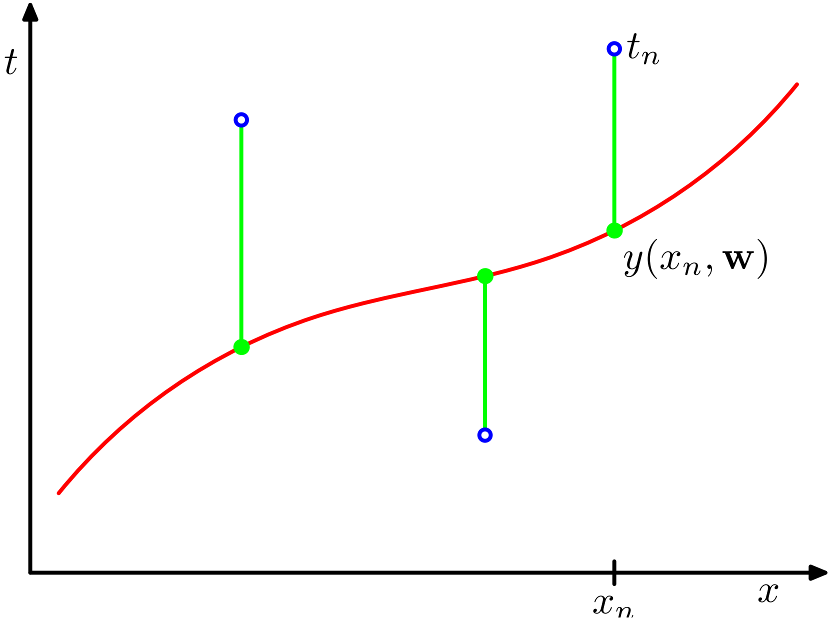


Figure 1.3 of Pattern Recognition and Machine Learning.

Linear Regression

Several ways how to minimize the error function – linear regression + sum of squares error have an explicit solution.

Our goal is to minimize:

$$\frac{1}{2}\sum_i^N (\boldsymbol{x}_i^T \boldsymbol{w} - t_i)^2.$$

If we denote $\boldsymbol{X} \in \mathbb{R}^{N\times D}$ the matrix of input values with $\boldsymbol{x}_i$ on a row $\boldsymbol{i}$ and $\boldsymbol{t} \in \mathbb{R}^N$ the vector of target values, we can it as

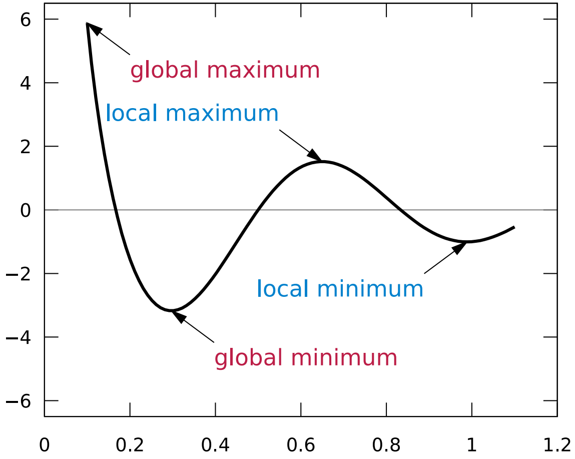
$$\frac{1}{2}\|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}\|^2,$$

because

$$\|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}\|^2 = \sum_i \big((\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})_i\big)^2 = \sum_i \big((\boldsymbol{X}\boldsymbol{w})_i - t_i)\big)^2 = \sum_i (\boldsymbol{x}_i^T \boldsymbol{w} - t_i)^2.$$

Minimization – Unconstrained, Single Real Variable

Assume we have a function and we want to find its minimum.



https://commons.wikimedia.org/wiki/File:Extrema_example_original.svg

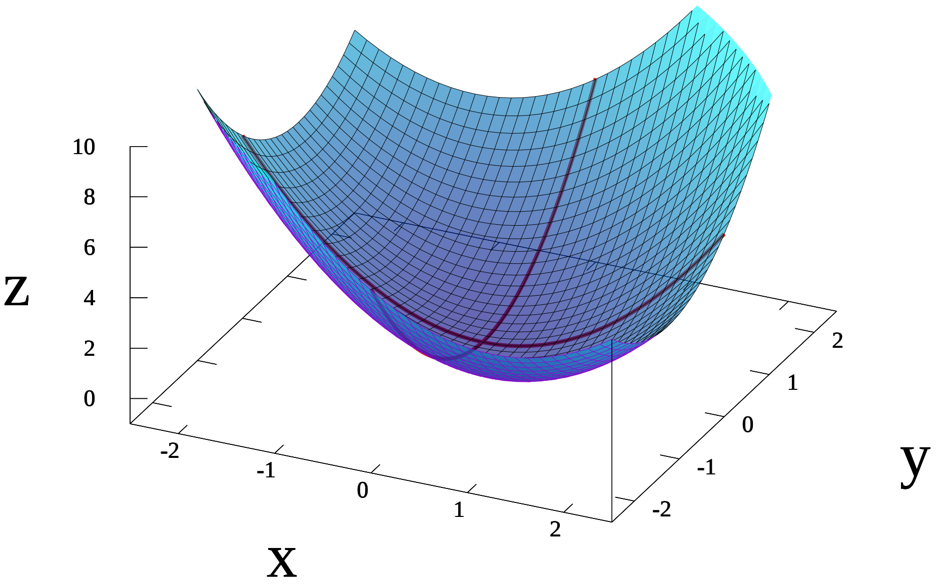
We usually use the Fermat's theorem (interior extremum theorem):

> Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be a function. If it has a minimum (or a maximum) in x and if it has a derivative in x, then $\frac{\partial f}{\partial x} = 0$.

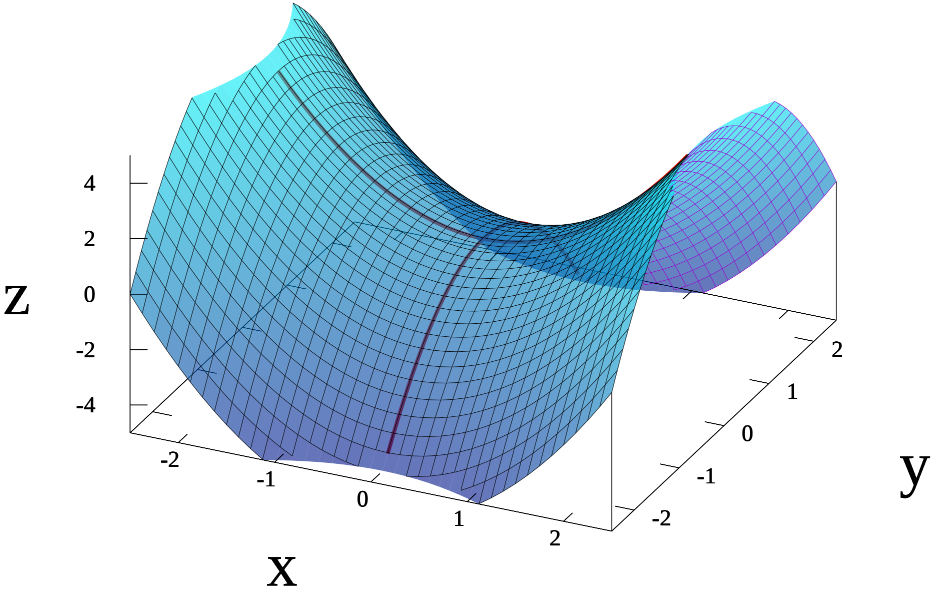
Minimization – Unconstrained, Multiple Real Variables

The previous theorem can be generalized to the multivariate case:

Let $f : \mathbb{R}^D \rightarrow \mathbb{R}$ be a function. If it has a minimum or a maximum in $\boldsymbol{x} = (x_1, x_2, \ldots, x_D)$ and if it has a derivative in $\boldsymbol{x}$, then for all i, $\frac{\partial f}{\partial x_i} = 0$. In other words, $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \mathbf{0}$.



https://commons.wikimedia.org/wiki/File:Partial_func_eg.svg



https://commons.wikimedia.org/wiki/File:Partial_func_eg.svg

Linear Regression

In order to find a minimum of $\frac{1}{2}\sum_i^N(\boldsymbol x_i^T\boldsymbol w - t_i)^2$, we can inspect values where the derivative of the error function is zero, with respect to all weights w_j.

$$\frac{\partial}{\partial w_j}\frac{1}{2}\sum_i^N(\boldsymbol x_i^T\boldsymbol w - t_i)^2 = \frac{1}{2}\sum_i^N\Big(2(\boldsymbol x_i^T\boldsymbol w - t_i)x_{ij}\Big) = \sum_i^N x_{ij}(\boldsymbol x_i^T\boldsymbol w - t_i)$$

Therefore, we want for all j that $\sum_i^N x_{ij}(\boldsymbol x_i^T\boldsymbol w - t_i) = 0$.

We can rewrite the explicit sum into $\boldsymbol X_{*,j}^T(\boldsymbol X\boldsymbol w - \boldsymbol t) = 0$, then write the equations for all j together using matrix notation as $\boldsymbol X^T(\boldsymbol X\boldsymbol w - \boldsymbol t) = \boldsymbol 0$, and finally, rewrite to

$$\boldsymbol X^T\boldsymbol X\boldsymbol w = \boldsymbol X^T\boldsymbol t.$$

The matrix $\boldsymbol X^T\boldsymbol X$ is of size $D\times D$. If it is regular, we can compute its inverse and therefore

$$\boldsymbol w = (\boldsymbol X^T\boldsymbol X)^{-1}\boldsymbol X^T\boldsymbol t.$$

Linear Regression

Input: Dataset ($\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \mathbb{R}^N$).
Output: Weights $\boldsymbol{w} \in \mathbb{R}^D$ minimizing MSE of linear regression.

- $\boldsymbol{w} \leftarrow (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}$.

The algorithm has complexity $\mathcal{O}(ND^2)$, assuming $N \geq D$.

When the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is singular, we can solve $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{t}$ using SVD, which will be demonstrated in the next lecture.

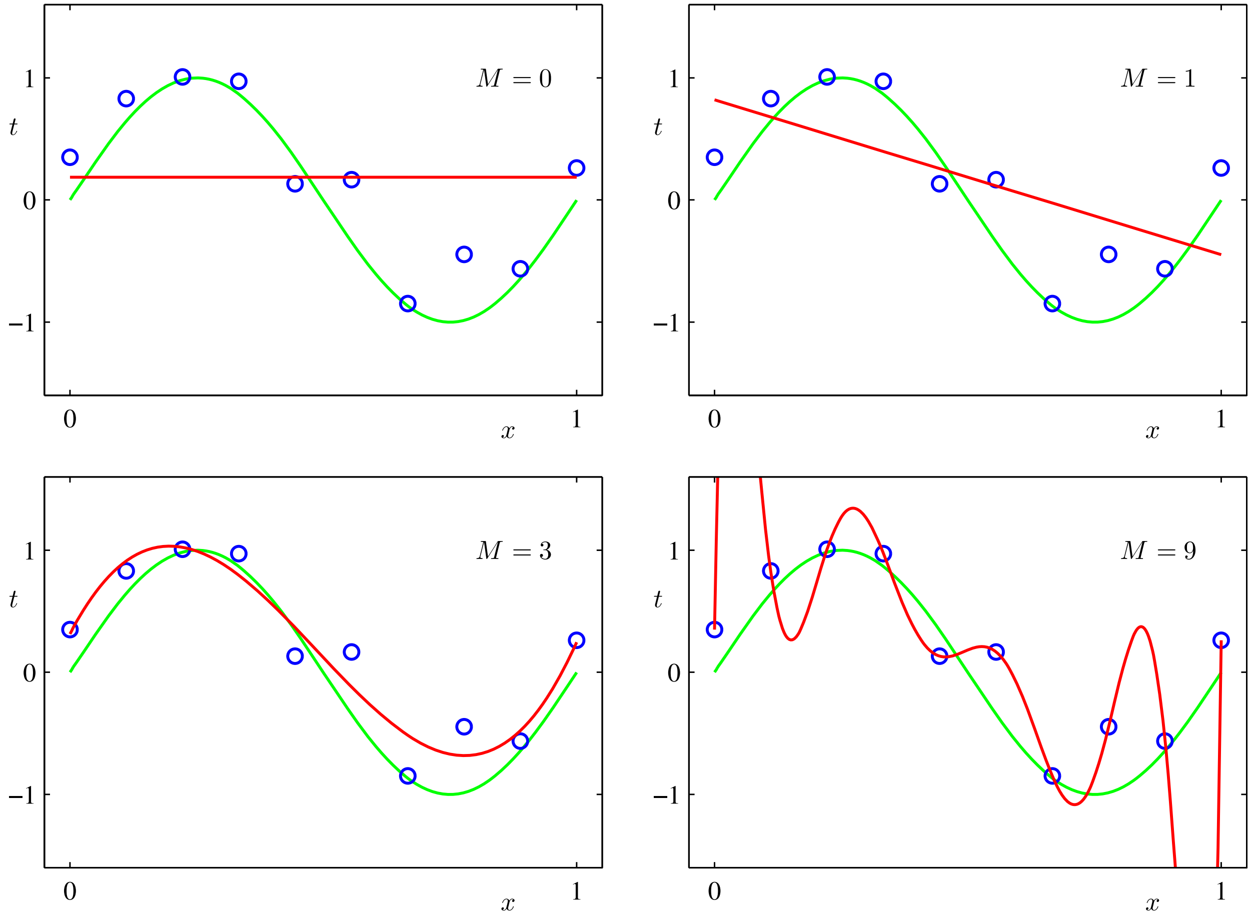
Linear Regression Example: Polynomial Features

We want to predict a $t \in \mathbb{R}$ for a given $x \in \mathbb{R}$. Linear regression with "raw" input vectors $\boldsymbol{x} = (x)$ can only model straight lines.

If we consider input vectors $\boldsymbol{x} = (x^0, x^1, \ldots, x^M)$ for a given $M \geq 0$, the linear regression is able to model polynomials of degree M. The prediction is then computed as

$$w_0 x^0 + w_1 x^1 + \ldots + w_M x^M.$$

The weights are the coefficients of a polynomial of degree M.



Linear Regression Example

To plot the error, the *root mean squared error* $\mathrm{RMSE} = \sqrt{\mathrm{MSE}}$ is frequently used.

The displayed error nicely illustrates two main challenges in machine learning:

- *underfitting*
- *overfitting*

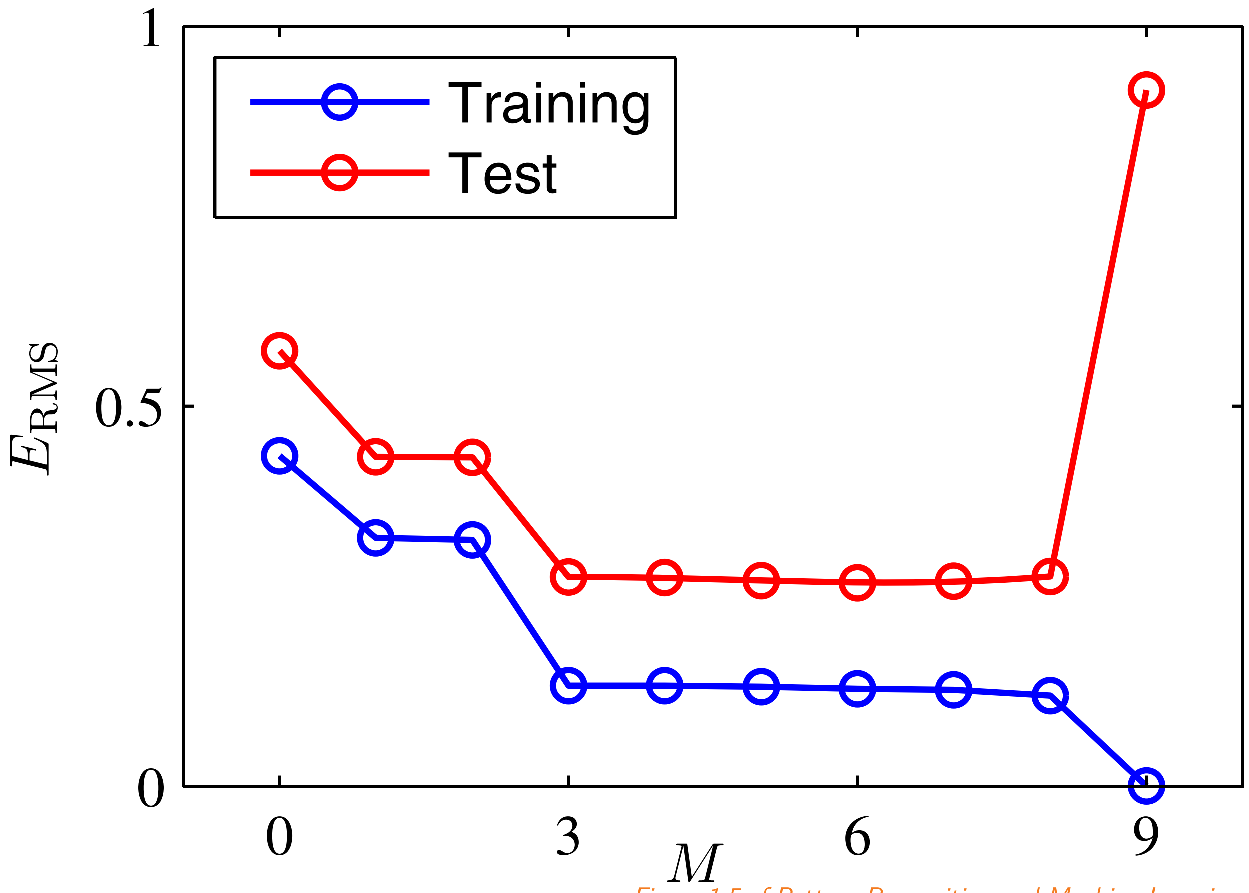


Figure 1.5 of Pattern Recognition and Machine Learning.

Today's Lecture Objectives

After this lecture you should be able to

- Explain to a non-expert what machine learning is
- Explain the difference between classification and regression
- Implement a simple linear-algebra-based algorithm for training linear regression

Reminder: Course Objectives

After this course you should...

- Be able to reason about task/problems **suitable for ML**
  - Know when to use classification, regression and clustering
  - Be able to choose from this method Linear and Logistic Regression, Multilayer Perceptron, Nearest Neighbors, Naive Bayes, Gradient Boosted Decision Trees, k-means clustering
- Think about learning as (mostly probabilistic) **optimization on training data**
  - Know how the ML methods learn including theoretical explanation
- Know how to properly **evaluate** ML
  - Think about generalization (and avoiding overfitting)
  - Be able to choose a suitable evaluation metric
  - Responsibly decide what model is better
- Be able to **implement ML algorithms** on a conceptual level
- Be able to **use Scikit-learn** to solve ML problems in Python