# IB031 Úvod do strojového učení

Tomáš Brázdil

# Course Info

Resources:

- Lectures & tutorials (the **main** source)
- Many books, few perfect for introductory level
  One relatively good, especially the first part:
  A. Géron. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media; 3rd edition, 2022
- (Almost) infinitely many online courses, tutorials, materials, etc.

# Evaluation

The evaluation is composed of three parts:

- Mid-term exam: Written exam from the material of the first half of the semester.
- End-term exam: The "big" one containing everything from the semester (with possibly more stress in the second half).
- Projects: During tutorials, you will work on larger projects (in pairs).

Each part contributes the following number of points:

- Mid-term exam: 25
- End-term exam: 50
- Project: 25

To pass, you need to obtain at least 60 points.

# Distinguishing Properties of the Course

- Introductory, prerequisites are held to a minimum
- Formal and precise: Be prepared for a complete and "mathematical" description of presented methods.

I assume that you have basic knowledge of

- Elementary understanding of mathematical notation (operations on sets, logic, etc.)
- Linear algebra: Vectors in $\mathbb{R}^n$, operations on vectors (including the dot product). Geometric interpretation!
- Calculus: Functions of multiple real variables, partial derivatives, basic differential calculus.
- Probability: Notion of probability distribution, random variables/vectors, expectation.

# What Is Machine Learning?

Machine learning is the science (and art) of programming computers so they can learn from data.

Here is a slightly more general definition:

### Arthur Samuel, 1959

Machine learning is the field of study that allows computers to learn without being explicitly programmed.

And a more engineering-oriented one:

### Tom Mitchell, 1997

A computer program is said to learn from experience $E$ concerning some task $T$ and some performance measure $P$ if its performance on $T$, as measured by $P$, improves with experience $E$.

# Example

In the context of spam filtering:

- The task $T$ is to flag spam in new emails.
- The experience $E$ is represented by a set of emails labeled either spam or ham by hand (the training data).
- The performance measure $P$ could be the accuracy, which is the ratio of the number of correctly classified emails and all emails.

  There are many more performance measures; we will study the basic ones later.

In the context of housing price prediction:

- The task $T$ is to predict prices of new houses based on their basic parameters (size, number of bathrooms, etc.)
- The experience $E$ is represented by information about existing houses.
- The performance measure $P$ could be, e.g., an absolute difference between the predicted and real price.

# Examples (cont.)

In the context of game playing:

- The task $T$ is to play chess.
- The experience $E$ is represented by a series of self-plays where the computer plays against itself.
- The performance measure $P$ is winning/losing the game.

  Here, the trick is to spread the delayed and limited feedback about the result of the game throughout the individual decisions in the game.

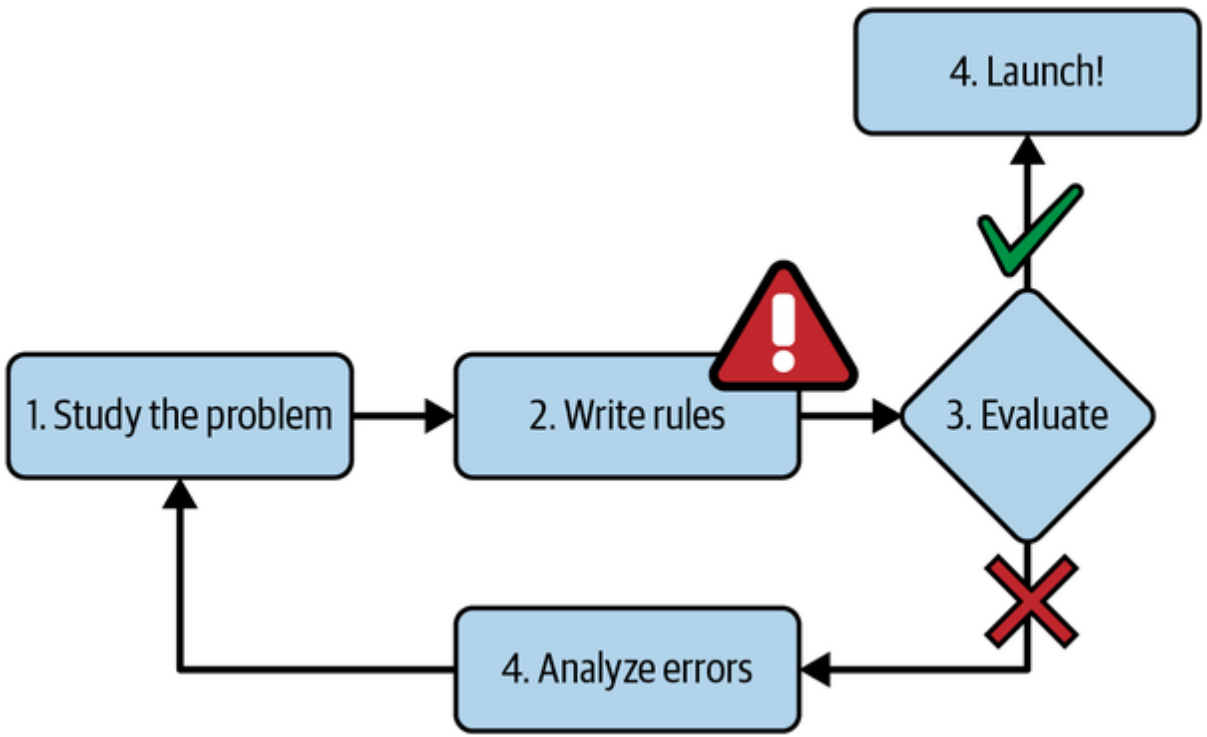
In the context of customer behavior:

- The task $T$ is to group customers with similar shopping habits in an e-shop.
- The experience $E$ consists of lists of items individual customers bought in the shop.
- The performance measure $P$?
  Measure how "nicely" the customers are grouped.
  (whether people with similar habits, as seen by humans, fall into the same group).

# Comparison of Programming and Learning

How to code the spam filter?

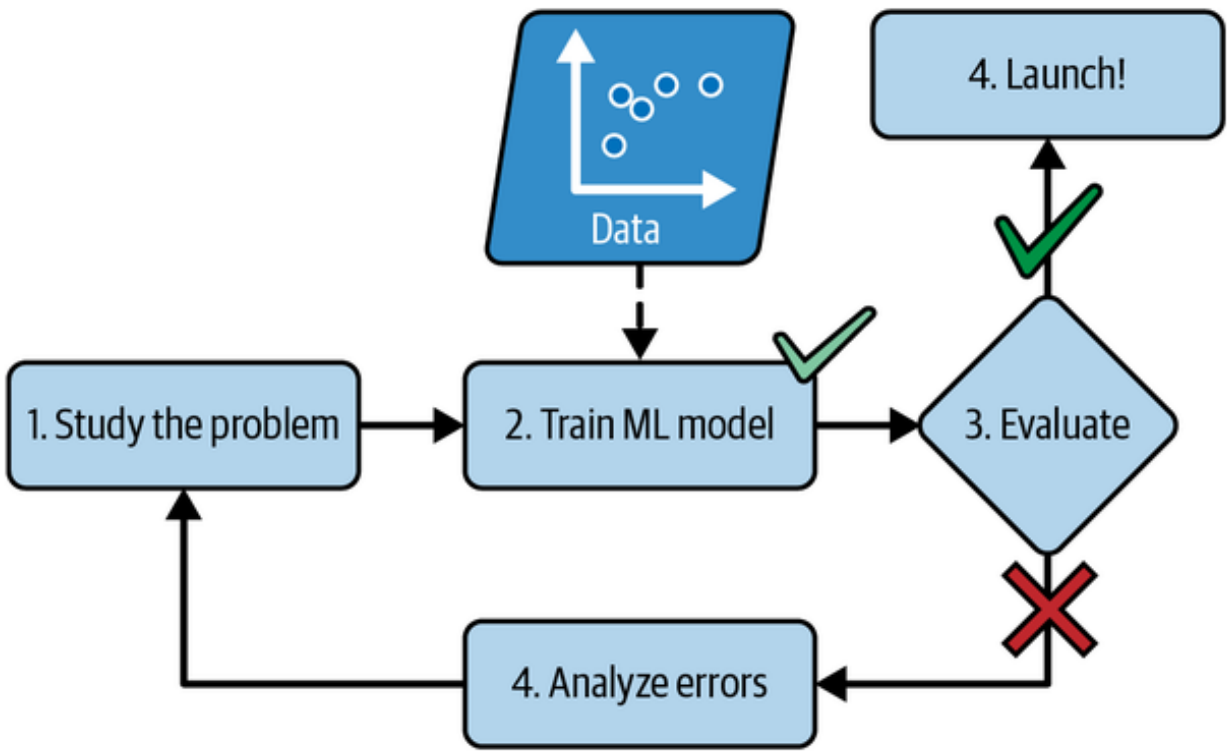
- Examine what spam mails typically contain: Specific words ("Viagra"), sender's address, etc.
- Write down a rule-based system that detects specific features.
- Test the program on new emails and (most probably) go back to look for more spam features.



# Comparison of Programming and Learning

The machine learning way:

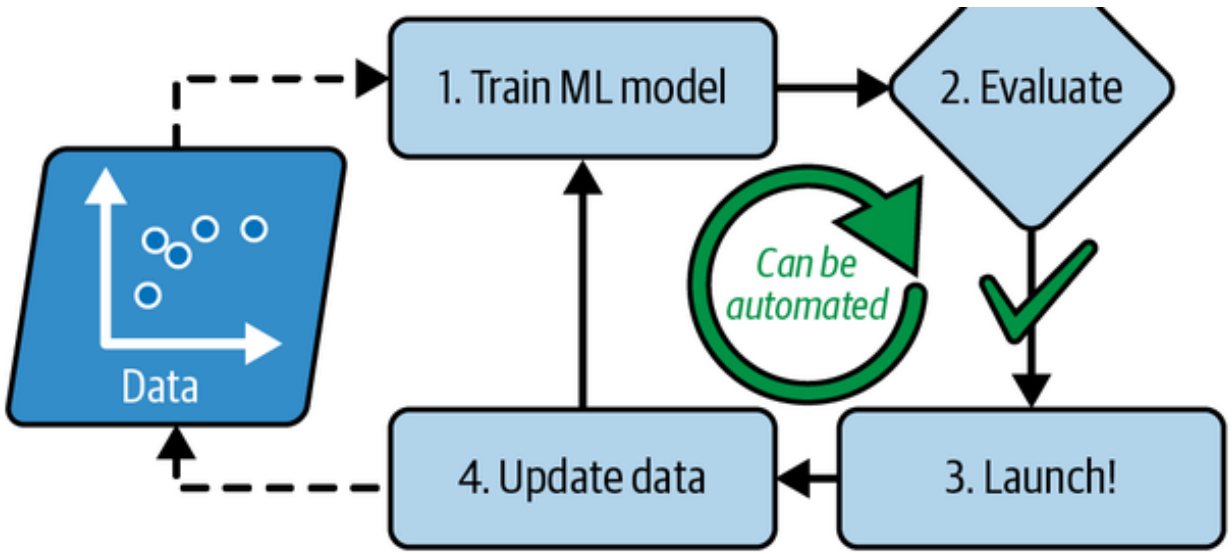
- Study the problem and collect lots of emails, labeling them spam or ham.
- Train a machine learning model that reads an email and decides whether it's spam or ham.
- Test the model and (most probably) go back to collect more data and adjust the model.



# ML Solutions are Adaptive

Spam filter: Authors of spam might and will adapt to your spam filter (possibly change the wording to pass through).

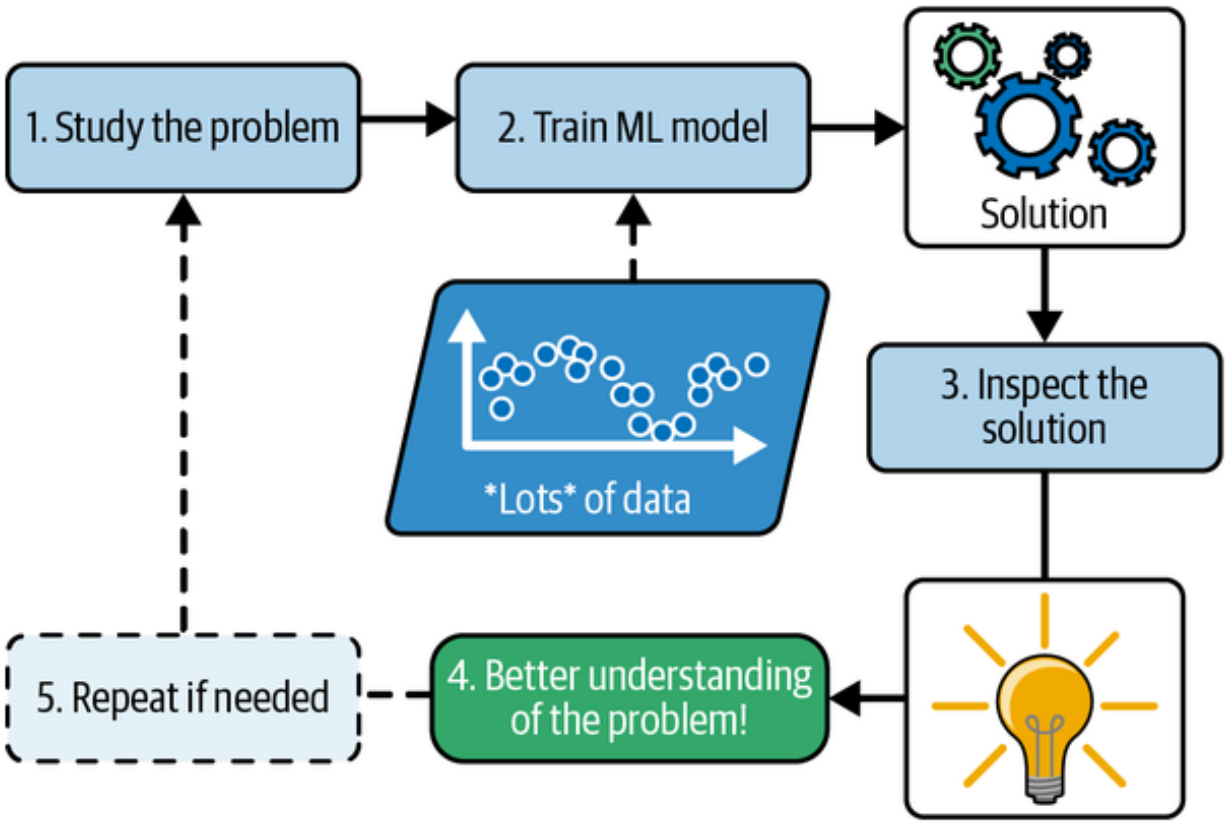
ML systems can be adjusted to new situations by retraining on new data (unless the data becomes ugly).



# ML for Human Understanding

Spam filter: A trained system can be inspected for notorious spam features.

Some models allow direct inspection, such as decision trees or linear/logistic regression models.



# Usage of Machine Learning

Machine learning suits various applications, especially where traditional methods fall short. Here are some areas where it excels:

- Solving complex problems where fine-tuning and rule-based solutions are inadequate.
- Tackling complex issues that resist traditional problem-solving approaches.
- Adapting to fluctuating environments through retraining on new data.
- Gaining insights from large and complex datasets.

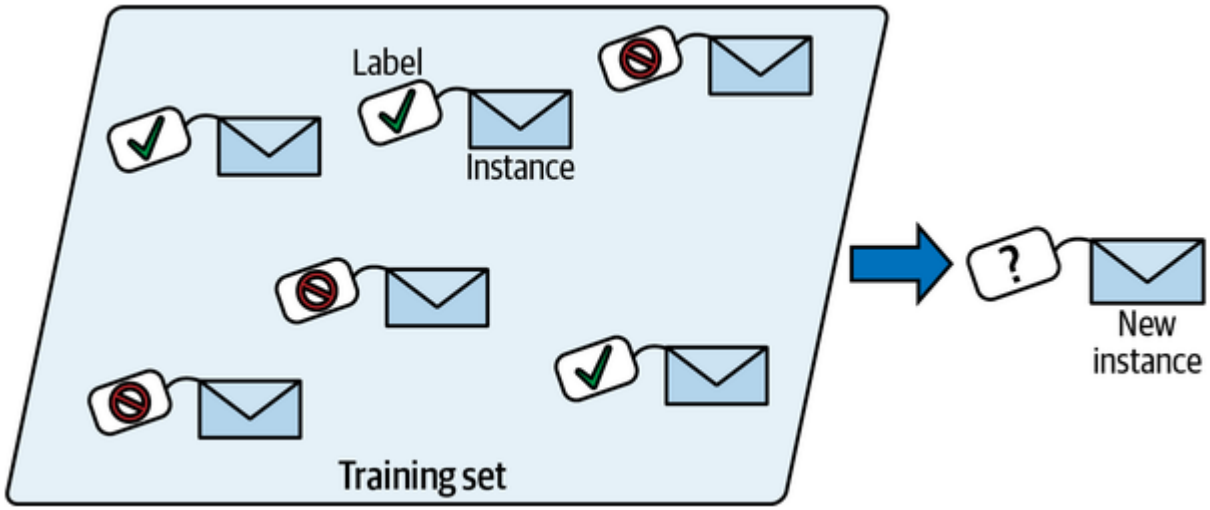
In summary, machine learning offers innovative solutions and adaptability for today's complex and ever-changing problems, (sometimes) providing insights beyond the reach of traditional approaches.

# Types of Learning

There are main categories based on information available during the training:

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Self-supervised learning
- Reinforcement learning

# Supervised Learning

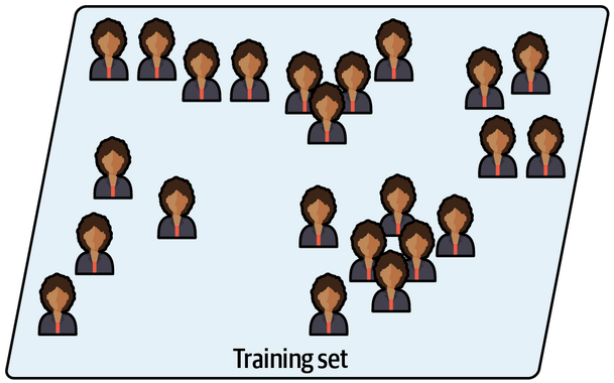


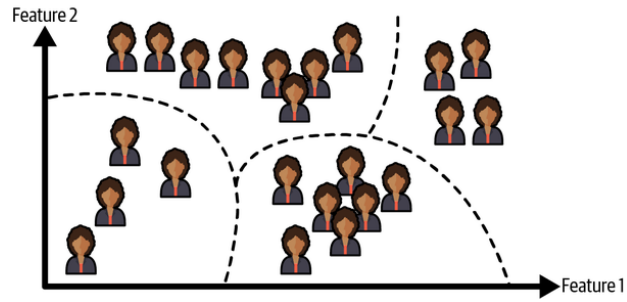

Labels are available for all input data.

Typical supervised learning tasks are

- *Classification* where the aim is to classify inputs into (typically few) classes
  (e.g., the spam filter where the classes are spam/ham)
- *Regression* where a numerical value is output for a given input
  (e.g., housing prices)

# Unsupervised Learning



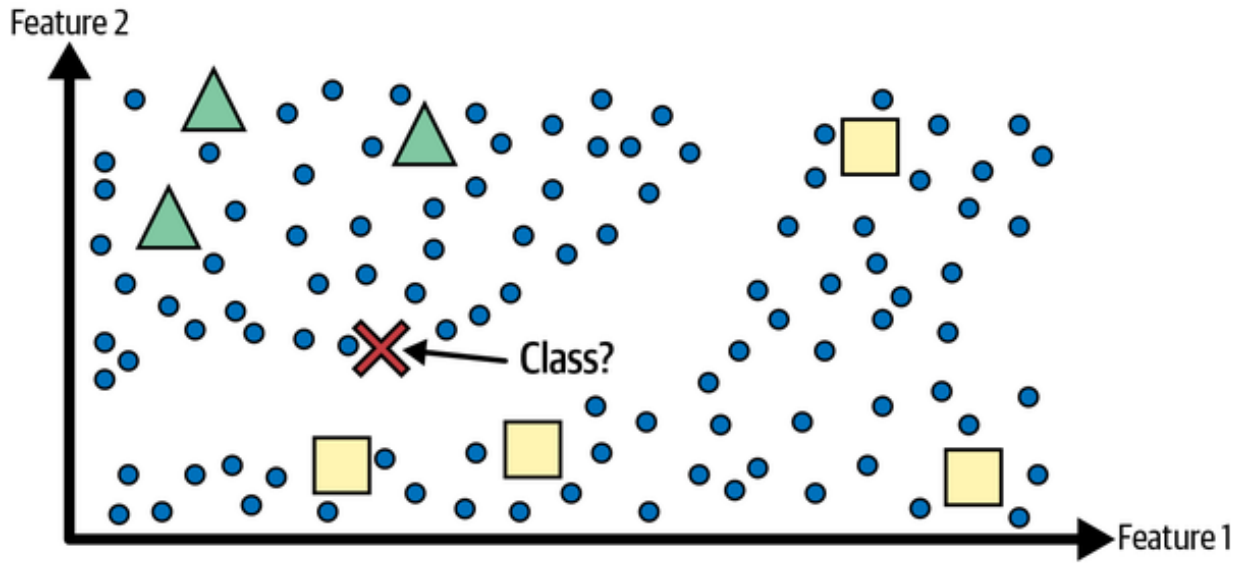




No labels are available for input data.

Typical unsupervised learning tasks are

- *Clustering* where inputs are grouped according to their features
  (e.g., clients of a bank grouped according to their age, wealth, etc.)
- *Association* where interesting relations and rules are discovered among the features of inputs
  (e.g., market basket mining where associations between various types of goods are being learned from the behavior of customers)
- *Dimensionality reduction* reduce high-dimensional data to few dimensions (e.g., images to few image features)

# Semi-Supervised Learning

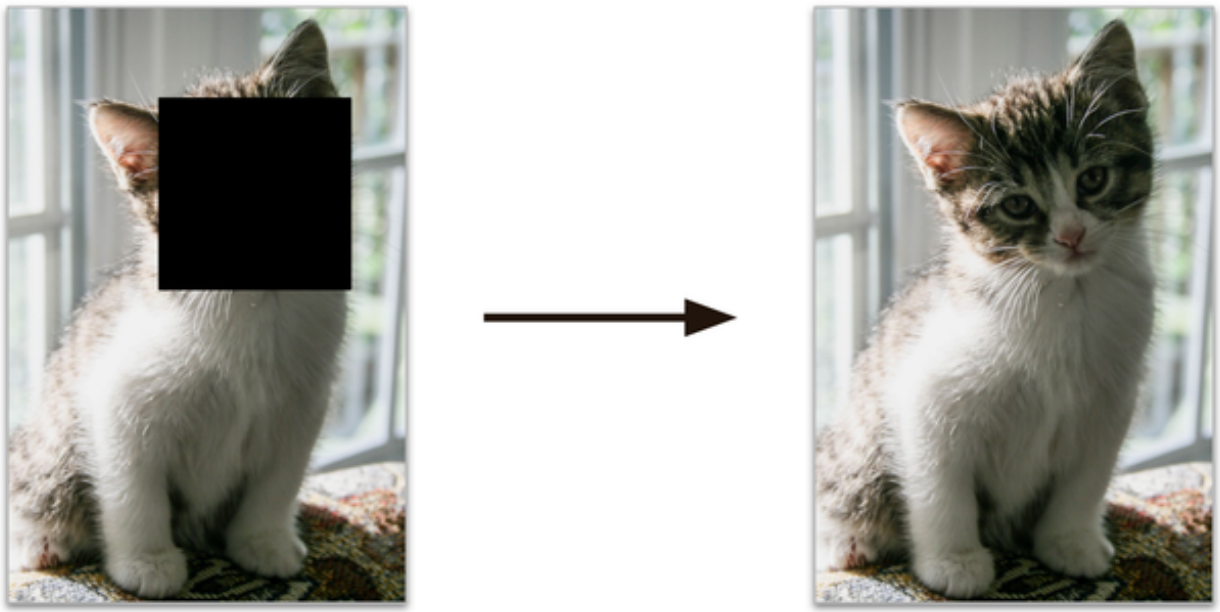



Labels for some data.

For example, Medical data, where elaborate diagnosis is available only for some patients.

Combines supervised and unsupervised learning: e.g., clusters all data and labels the unlabeled inputs with the most common labels in their clusters.

# Self-Supervised Learning

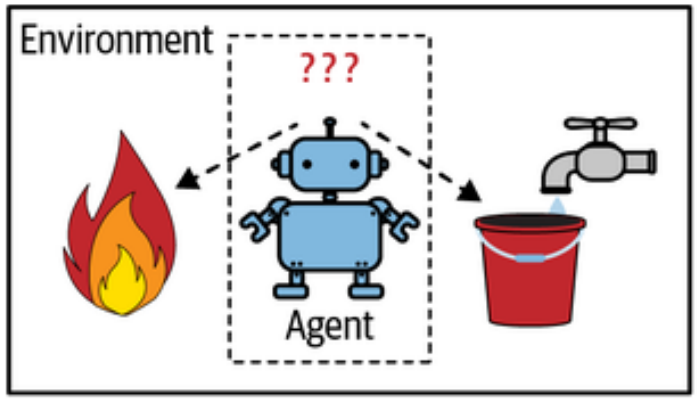


Generate labels from (unlabeled) inputs.

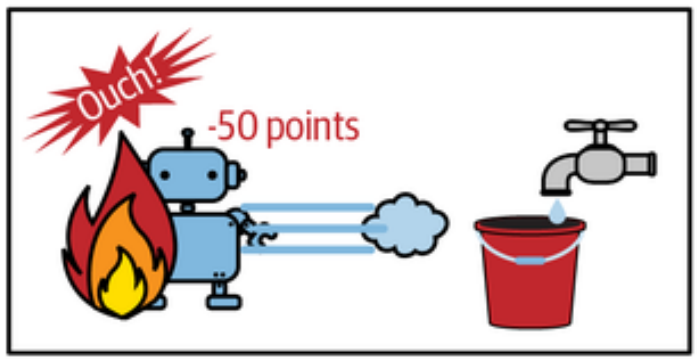
The goal is to learn typical features of the data.

It can be later modified to generate images, classify, etc.

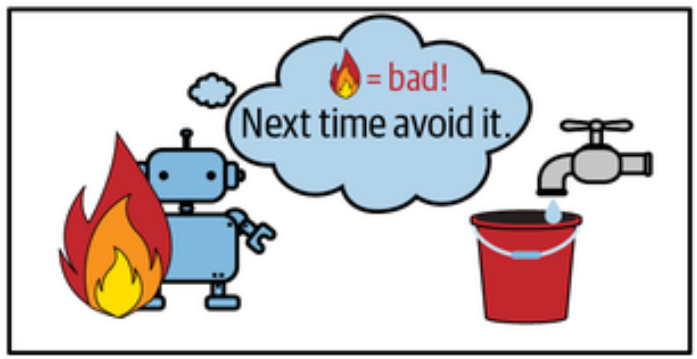
# Reinforcement Learning




1. Observe
2. Select action using policy




3. Action!
4. Get reward or penalty



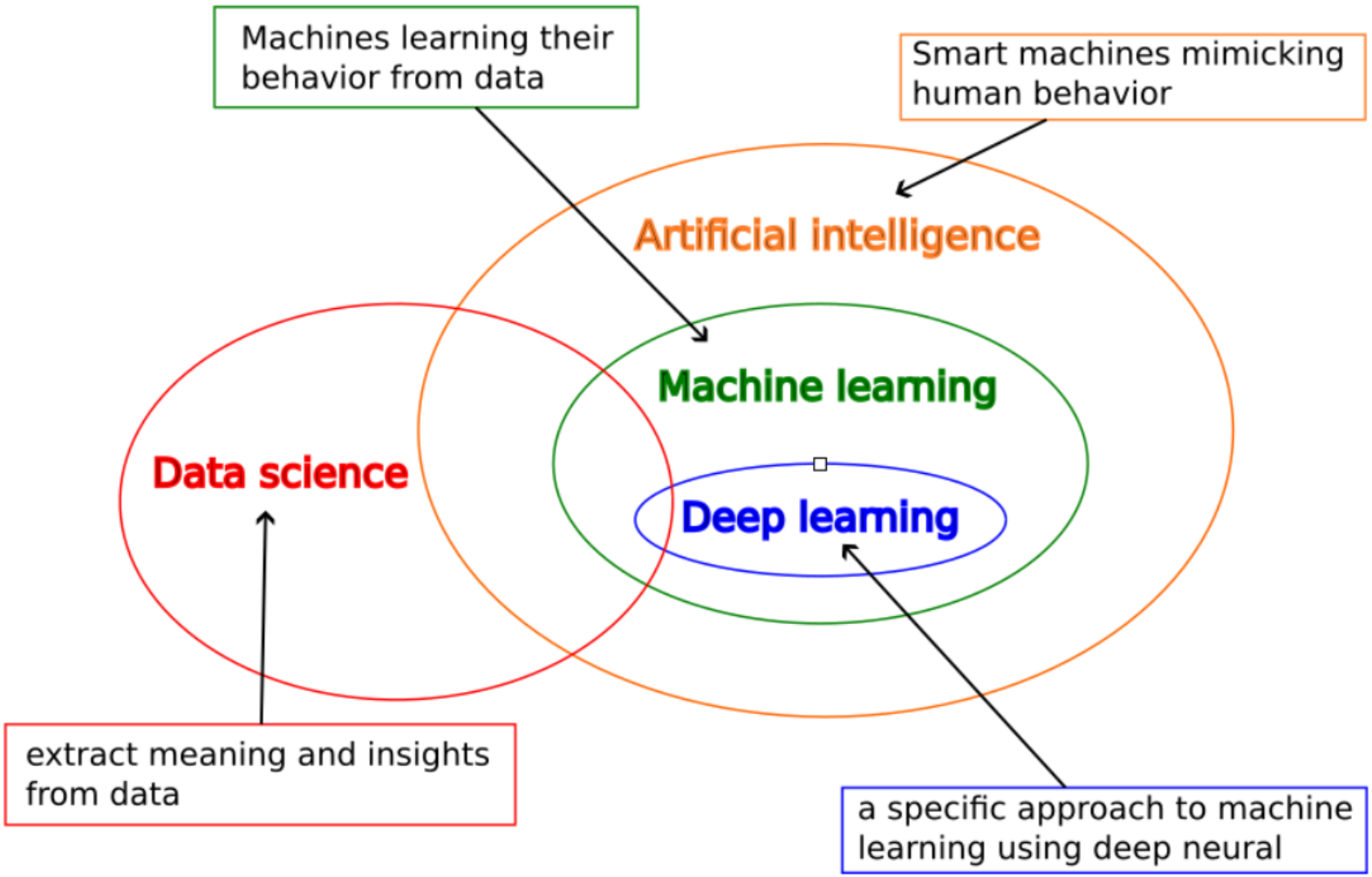

5. Update policy (learning step)
6. Iterate until an optimal policy is found

Learn from performing *actions* and getting feedback from *environment*.

# ML Applications Highlights

- ChatGPT (and similar generative models)
  - The basis forms a generative language model, i.e., a text-generating model trained on texts in a self-supervised way
  - Currently extended to multimodal versions (text, image, sound)
- Machine translation, image captioning
  - Google translate, etc.
  - Typically (semi)-supervised learning,
- Various image recognition and processing tasks
  - In medicine where it is slowly making its way into hospitals as assistance tools
  - Automotive, advertising, quality control etc., etc., etc.
- Science
  - Chemistry & biology: E.g., prediction of features of chemical compounds (Alpha-fold)
- Various "table" data processing in finance, management, etc.
  - Often straightforward methods (linear/logistic regression)
  - Essential but not fancy
- Game playing: More fancy than useful, learning models beating humans in several difficult games.

# ML in Context



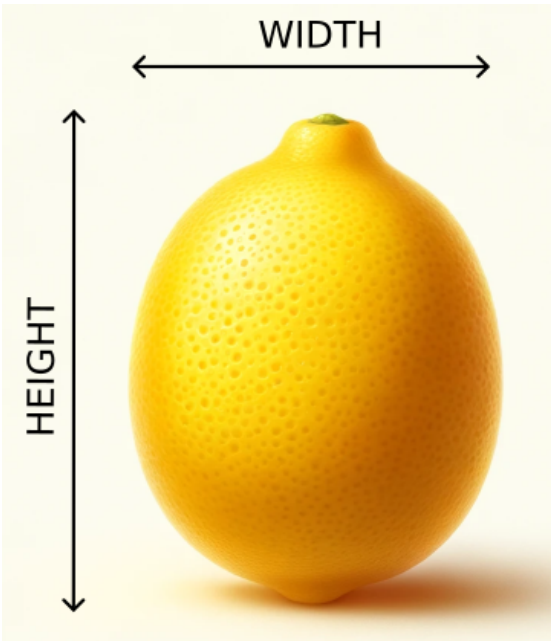
# Supervised Learning

# Example - Fruit Recognition

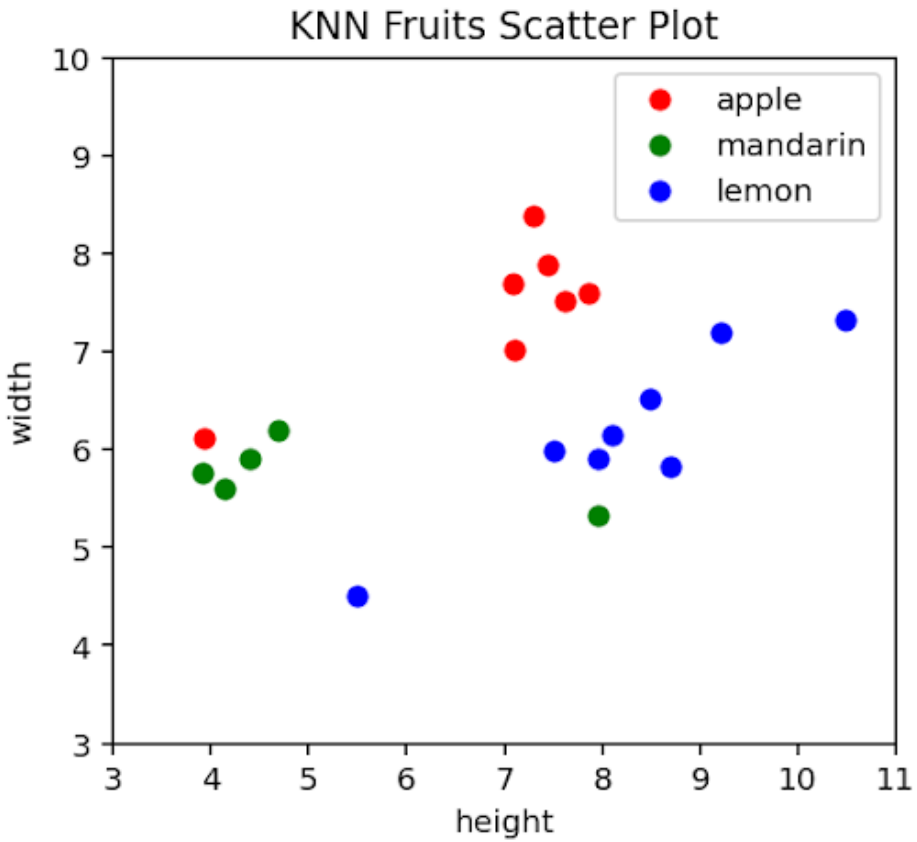
**The goal:** Create an automatic system for fruit recognition, concretely apple, lemon, and mandarin.

**Inputs:** Measures of *height* and *width* of each fruit.

Suppose we have a dataset of dimensions of several fruits labeled with the correct class.



# Data



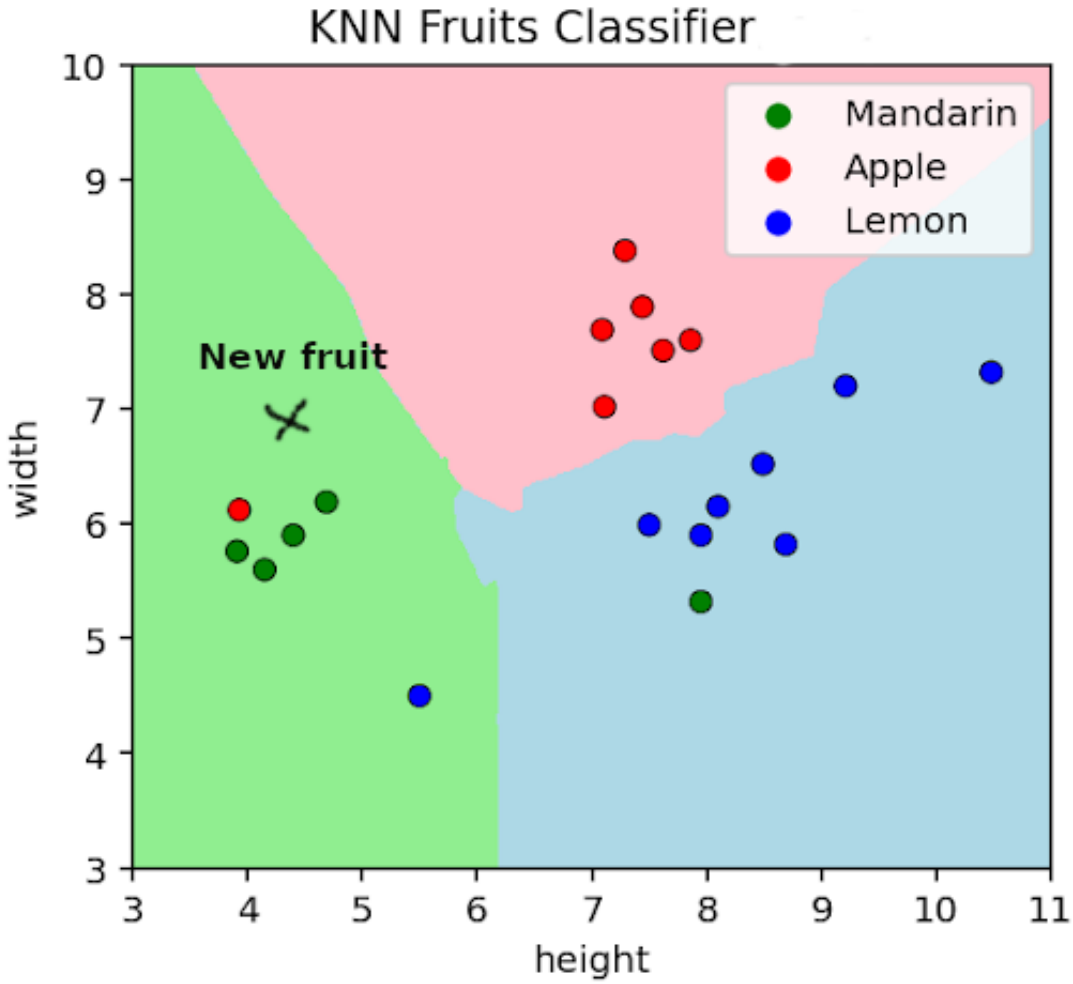

| | height | width | fruit |
|---|---|---|---|
| 0 | 3.91 | 5.76 | Mandarin |
| 1 | 7.09 | 7.69 | Apple |
| 2 | 10.48 | 7.32 | Lemon |
| 3 | 9.21 | 7.20 | Lemon |
| 4 | 7.95 | 5.90 | Lemon |
| 5 | 7.62 | 7.51 | Apple |
| 6 | 7.95 | 5.32 | Mandarin |
| 7 | 4.69 | 6.19 | Mandarin |
| 8 | 7.50 | 5.99 | Lemon |
| 9 | 7.11 | 7.02 | Apple |
| 10 | 4.15 | 5.60 | Mandarin |
| 11 | 7.29 | 8.38 | Apple |
| 12 | 8.49 | 6.52 | Lemon |
| 13 | 7.44 | 7.89 | Apple |
| 14 | 7.86 | 7.60 | Apple |
| 15 | 3.93 | 6.12 | Apple |
| 16 | 4.40 | 5.90 | Mandarin |
| 17 | 5.50 | 4.50 | Lemon |
| 18 | 8.10 | 6.15 | Lemon |
| 19 | 8.69 | 5.82 | Lemon |

Use similarity to solve the problem.

# KNN Classification

Given a new fruit.
What is it?

Find five closest examples




Where is the machine learning?

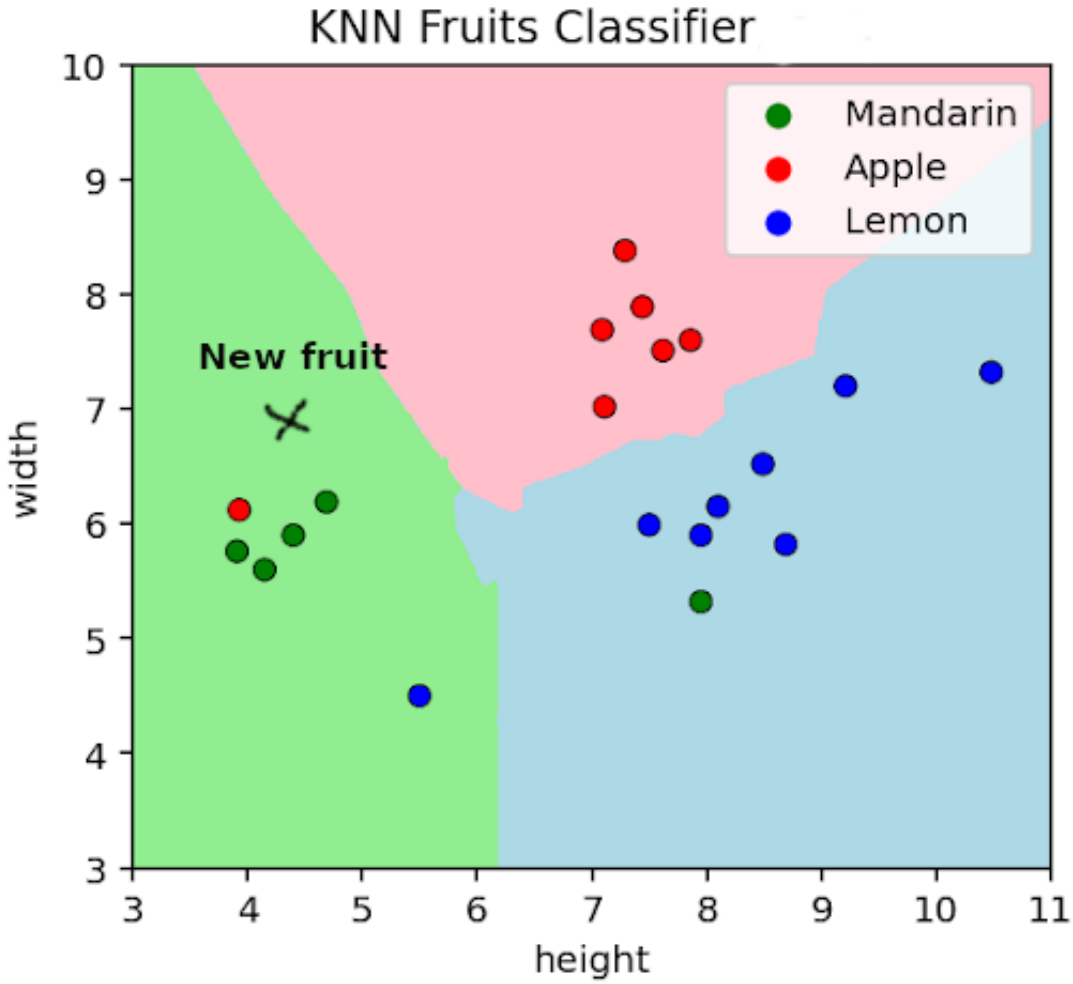
# KNN Classification

Given a new fruit.
What is it?

Find five closest examples

Among the five closest:

- $M = 4$ mandarins
- $A = 1$ apples
- $L = 0$ lemons




Where is the machine learning?

# KNN Classification

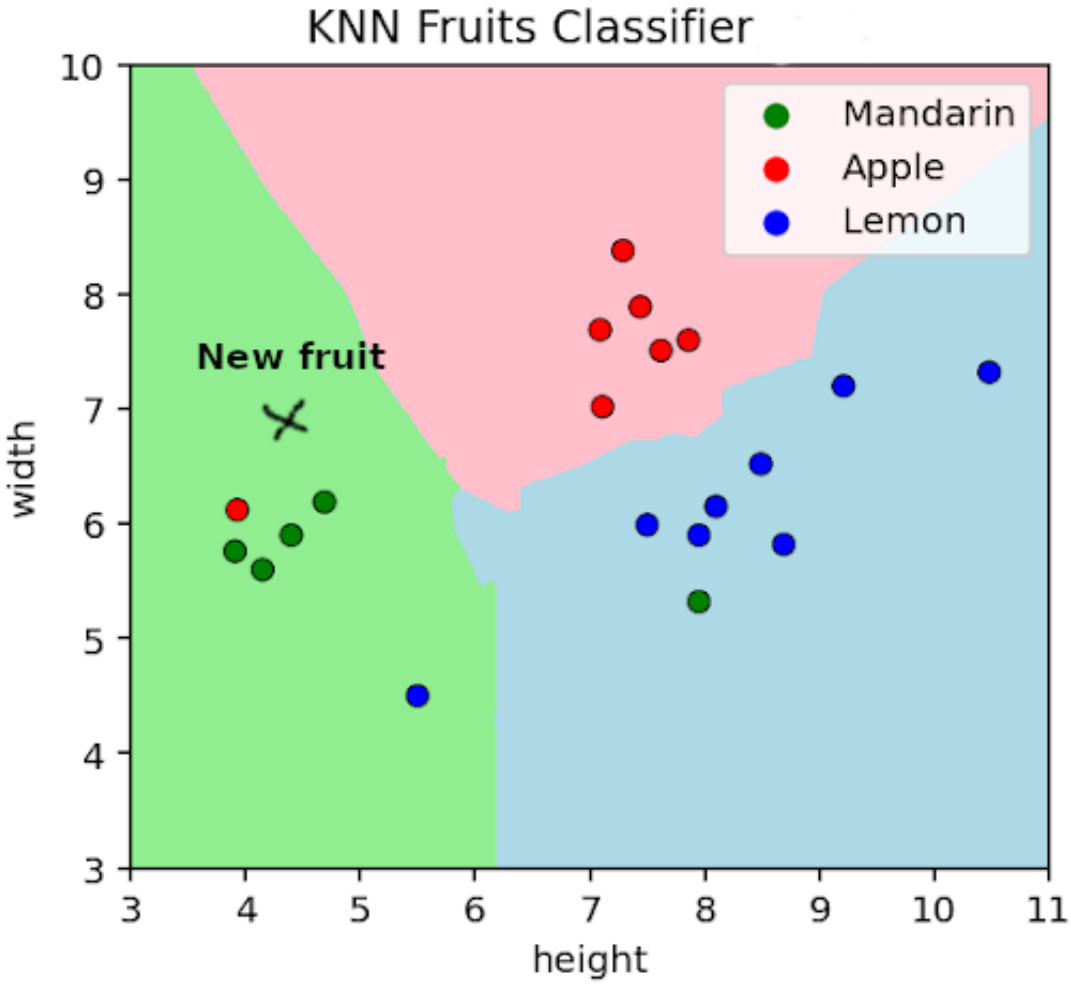
Given a new fruit.
What is it?

Find five closest examples

Among the five closest:

- $M = 4$ mandarins
- $A = 1$ apples
- $L = 0$ lemons

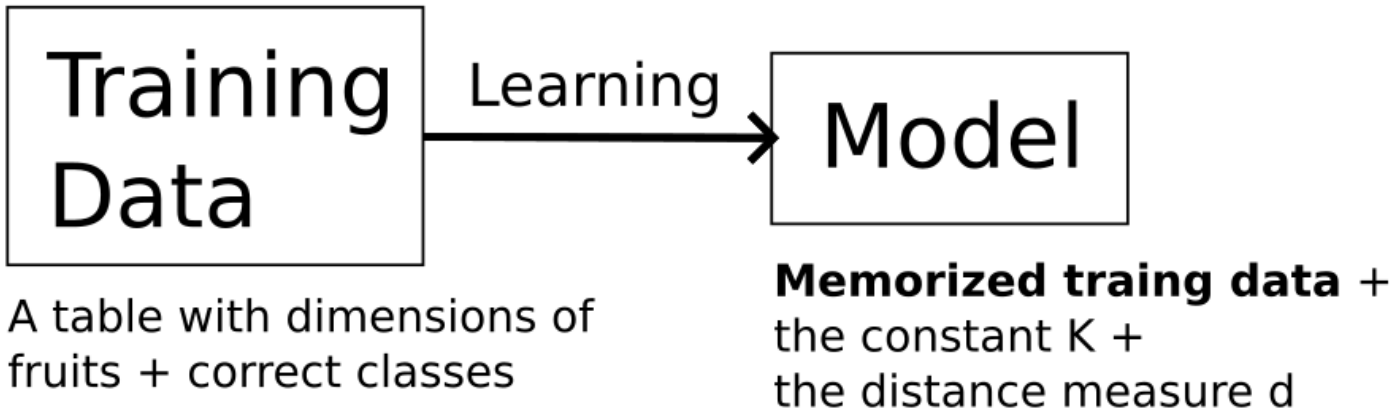
It is a **mandarin**!



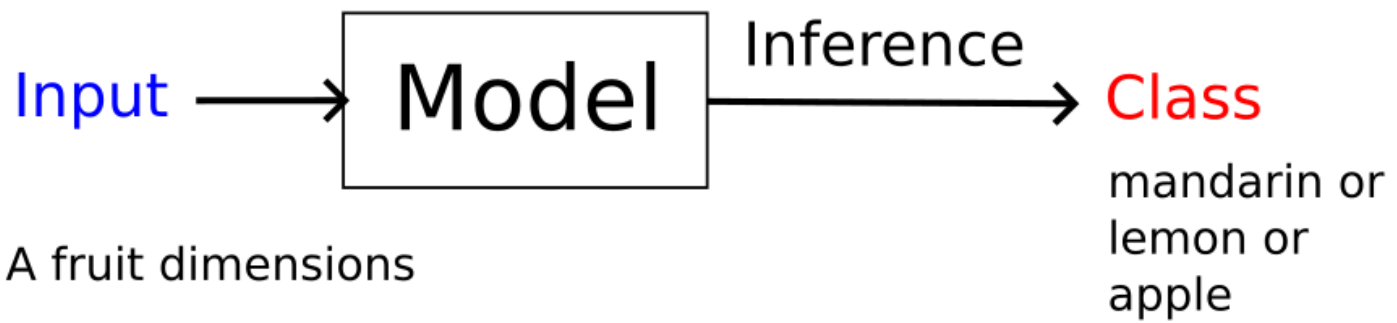

Where is the machine learning?

# Learning in Fruit Classification with KNN

**Learning:**




**Inference:**



# Fruit Classification Algorithm

**Input:** A fruit $F$ with dimensions *height*, *width*
**Output:** *mandarin*, *lemon*, *apple*

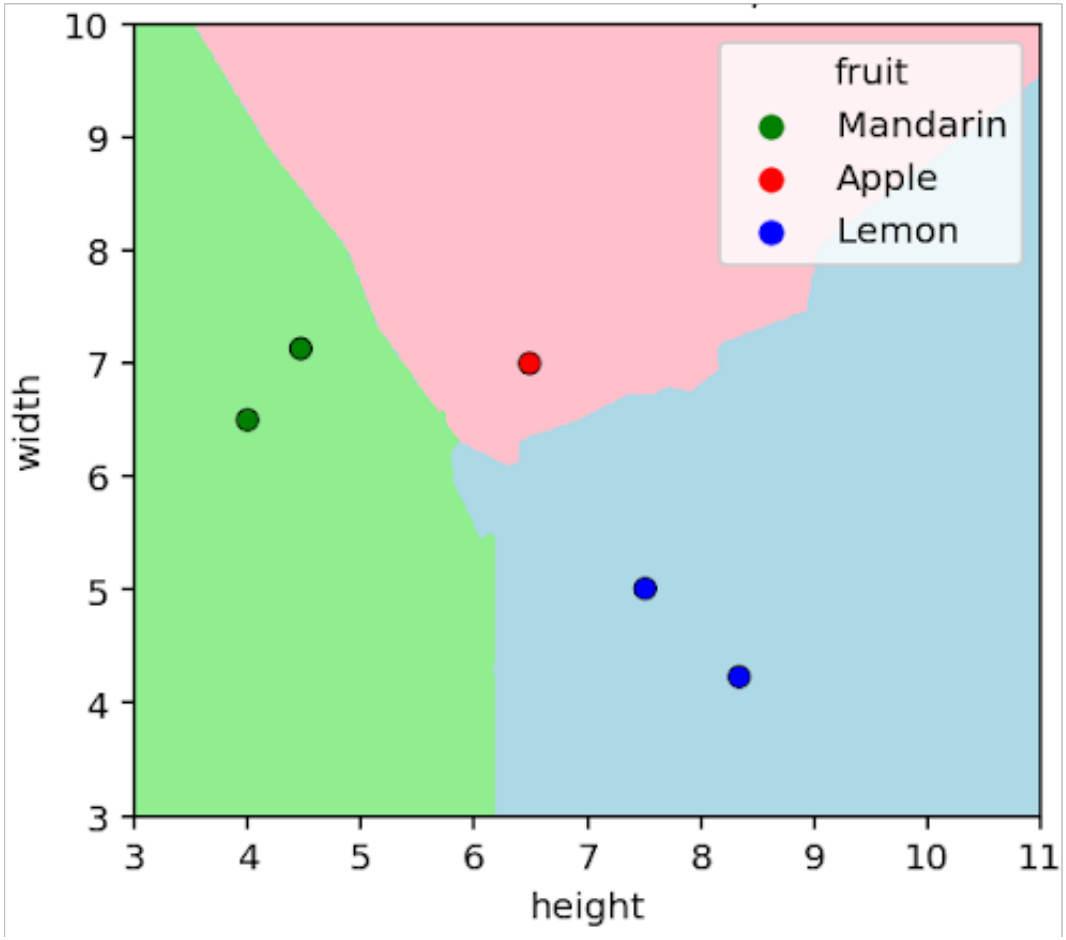
1: Find $K$ examples $\{E_1, \ldots, E_K\}$ in the dataset whose dimensions are closest to the dimensions of the fruit $F$
2: Count the number of examples of each class in $\{E_1, \ldots, E_K\}$
    $M$ mandarins in $\{E_1, \ldots, E_K\}$
    $L$ lemons in $\{E_1, \ldots, E_K\}$
    $A$ apples in $\{E_1, \ldots, E_K\}$
3: **if** $M \geq L$ and $M \geq A$ **then return** *mandarin*
4: **else if** $L \geq A$ **then return** *lemon*
5: **else return** *apple*
6: **end if**

Does it work?

# Testing the Model for Fruit Classification

Consider a test set of new instances ($K = 5$, $d$ is Euclidean):

| height | width | fruit |
|---|---|---|
| 4.0 | 6.5 | Mandarin |
| 4.47 | 7.13 | Mandarin |
| 6.49 | 7.0 | Apple |
| 7.51 | 5.01 | Lemon |
| 8.34 | 4.23 | Lemon |




Perfect classification of new data! Just deploy and sell!!

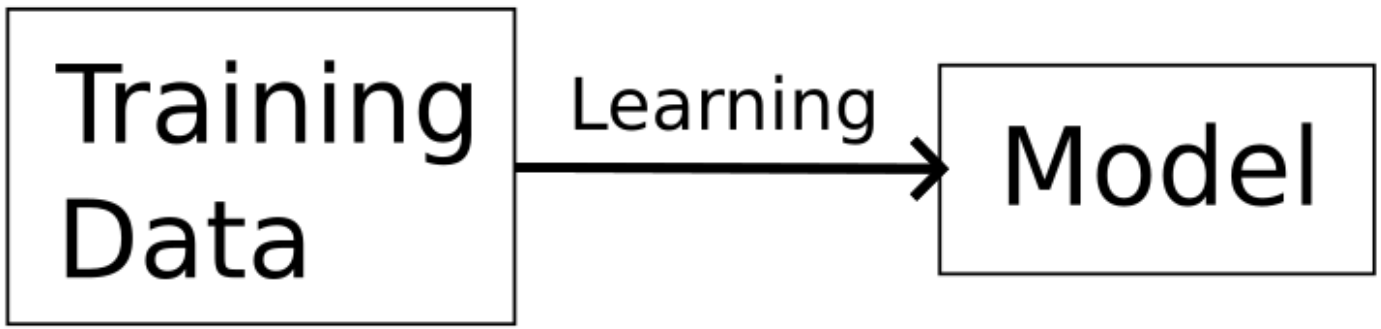
# K Nearest Neighbors

. . . on ideal data

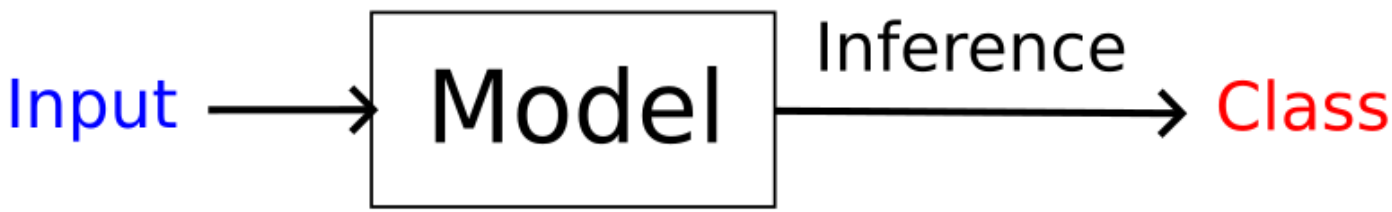
# Learning and Inference

Two crucial components of machine learning are the following:

**Learning:**
Creating model




**Inference:**
Using model



# Training Data

Assume table training data, i.e., of the form

$$\begin{array}{cccc|c} x_{11} & x_{12} & \cdots & x_{1n} & c_1 \\ x_{21} & x_{22} & \cdots & x_{2n} & c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{p1} & x_{p2} & \cdots & x_{pn} & c_p \end{array}$$

| height | width | fruit |
|---|---|---|
| 4.0 | 6.5 | Mandarin |
| 4.47 | 7.13 | Mandarin |
| 6.49 | 7.0 | Apple |
| 7.51 | 5.01 | Lemon |
| 8.34 | 4.23 | Lemon |

Formally, we define **training dataset**

$$\mathcal{T} = \{(\vec{x}_k, c_k) \mid k = 1, \ldots, p\}$$

Here each $\vec{x}_k \in \mathbb{R}^n$ is an input vector and $c_k \in C$ is the correct class.

$$\mathcal{T} = \{(4.0, 6.5), M), (4.47, 7.13), M), (6.49, 7.0), A), \ldots\}$$

# KNN: Learning

Consider the training set:

$$\mathcal{T} = \{(\vec{x}_k, c_k) \mid k = 1, \ldots, p\}$$

and *memorize it* exactly as it is.

Store in a table.

Possibly use a clever representation allowing fast computation of nearest neighbors such as KDTrees (out of the scope of this lecture).

Also,

- determine the number of neighbors $K \in \mathbb{N}$,
- and the distance measure $d$.

# Inference in KNN

Assume a KNN "trained" by memorizing $\mathcal{T} = \{(\vec{x}_k, c_k) \in \mathbb{R}^n \times C \mid k = 1, \ldots, p\}$, a constant $K \in \mathbb{N}$ and a distance measure $d$.

For $d$, consider Euclidean distance, but different norms may also be used to define different distance measures.

**Input:** A vector $\vec{z} = (z_1, \ldots, z_n) \in \mathbb{R}^n$

**Output:** A class from $C$

1: Find $K$ indices of examples $X = \{i_1, \ldots, i_K\} \subseteq \{1, \ldots, p\}$ with minimum distance to $\vec{z}$, i.e., satisfying

$$\max\Big\{d(\vec{z}, \vec{x}_\ell) \mid \ell \in X\Big\} \leq \min\Big\{d(\vec{z}, \vec{x}_\ell) \mid \ell \in \{1, \ldots, p\} \setminus X\Big\}$$

2: For every $c \in C$ count the number $\#c$ of elements $\ell$ in $X$ such that $c_\ell = c$

3: Return some

$$c_{max} \in \arg\max_{c \in C} \#c$$

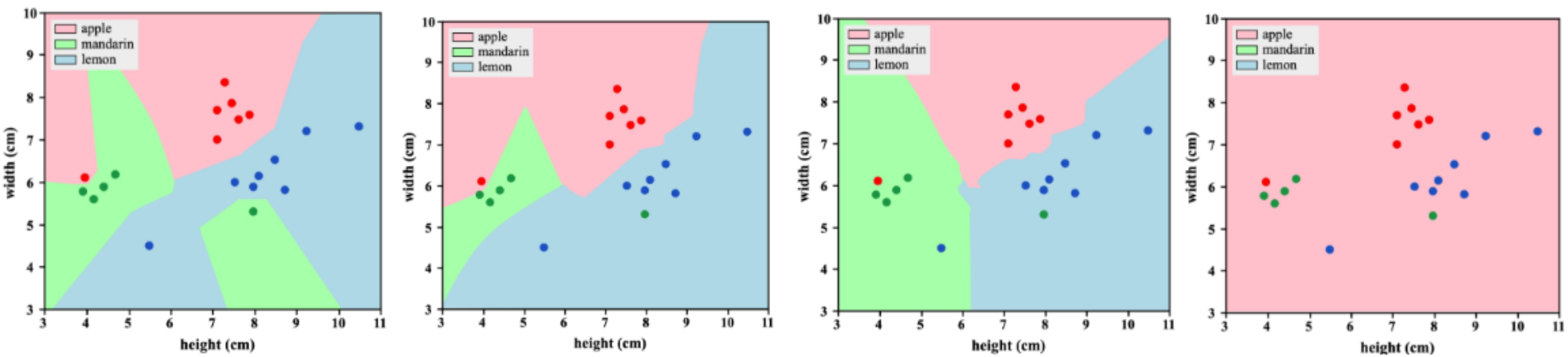
A class $c_{max} \in C$ which maximizes $\#c$.

# The resulting model

What exactly constitutes the model? The *model* consists of

- The *trained parameters*: In this case the memorized training data.
- The *hyperparameters* set "from the outside": In this case, the number of neighbors $K$ and the distance measure $d$.

Note that different settings of $K$ lead to different classifiers (for the same $d$):



# In Practice

. . . to get an efficient solution:

- Deal with issues in the data
  - Data almost always comes in weird formats, with inconsistencies, missing values, wrong values, etc.
  - Data rarely have the ideal form for a given learning model.

  We need to ingest, validate, and preprocess the data.
- Deal with issues in the model
  - In KNN, the training memorizes the example, but at least the $K$ can be tuned.

  We need to tune the model.
- Deal with the wrong model by testing and validation in as realistic conditions as possible.
- Deal with deployment - real-world application issues involving, e.g., implementation in embedded devices with limited resources.

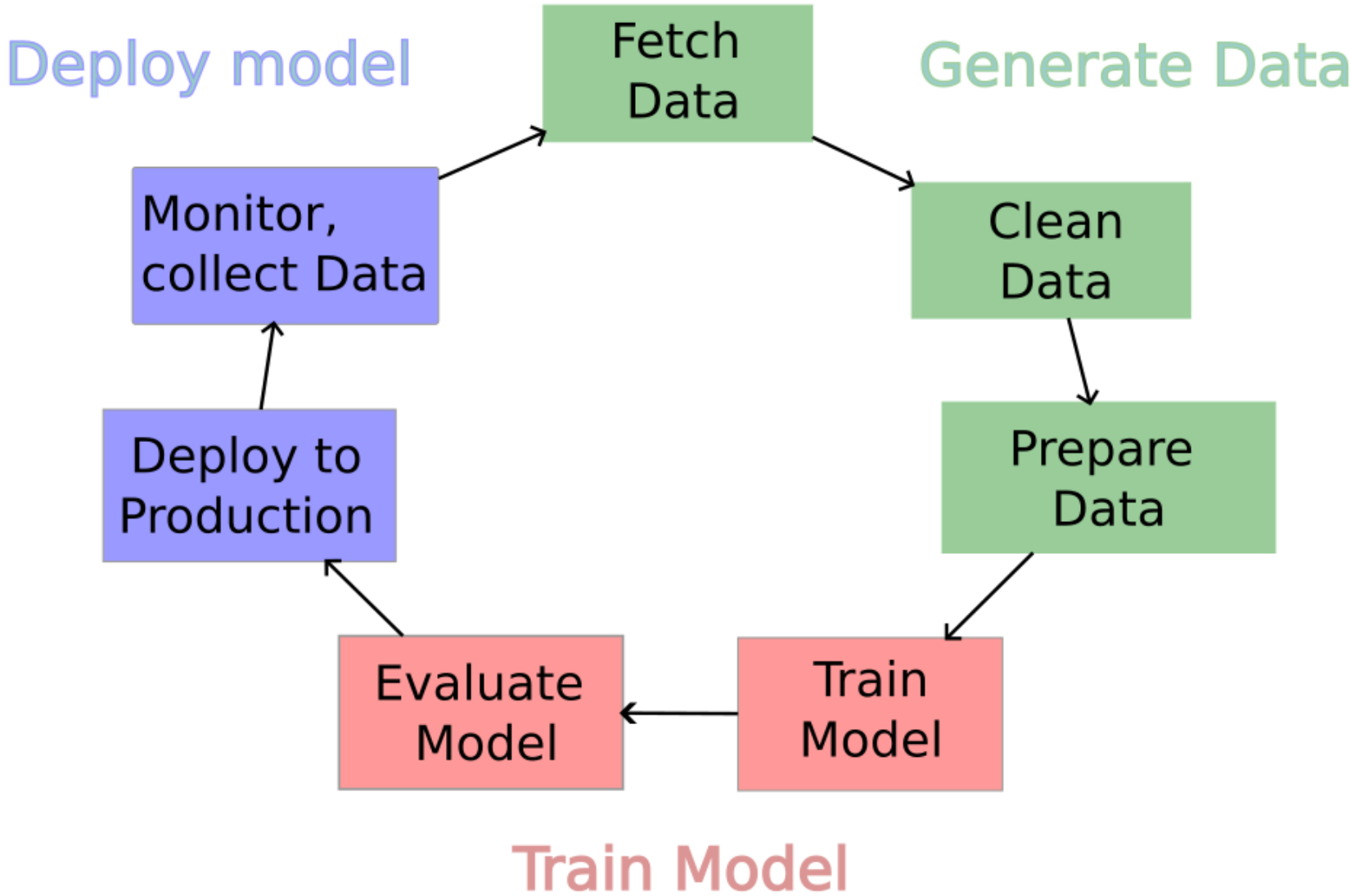
# Models Considered in This Course

Throughout this course, we will meet the following models:

- KNN (already did)
- Decision trees
- (Naive) Bayes classifier
- Clustering: K-means and hierarchical
- Linear and logistic regression
- Support Vector Machines (SVM)
- Kernel linear models
- Neural networks (light intro to feed-forward networks)
- Ensemble methods + random forests
- (maybe some reinforcement learning)

. . . but first, let us see the whole machine learning pipeline.

# Machine Learning Pipeline



# Fetch Data

Always start with

- The problem formulation & understanding.

  For example, diagnosis of diabetes from medical records. What info is (possibly) sufficient for such a diagnosis?

- Find data sources.

  In our example, the sources are hospitals. It would be best to persuade them to give you the data and sign a contract.

- Collect the data.

  In our example, the data is possibly small (just tables with results of tests). But for other diagnoses, you may include huge amounts of data from MRI, CT, etc. Then, the collection itself might be a serious technical problem.

- Integrate data from various sources.

  A serious diagnostic system must be trained/tested on data from many hospitals. You must blend the data from various sources (different formats, etc.).

# Fetch Data

For simple "toy" machine learning projects, you may fetch prepared datasets from various databases on the internet.

The data should be stored in an identified location and versioned.
You will probably keep adding data and training models on the ever-changing datasets. You have to be able to keep track of the changes and map training data to particular models.

Tools such as ML Flow or Weights & Biases might be helpful.

**Data Separation**
At this point, you should randomize the ordering of the data and select a test set to be used in model evaluation!

The test data are supposed to simulate the actual conditions, i.e., they should be "unseen".

**Data Exploration**
Compute basic statistics to identify missing values, outliers, etc.

# Clean Data

The cleaning usually comprises the following steps:

- Fix or remove incorrect or corrupted values.
- Identify outliers and decide what to do with them.

  Outliers may harm some training methods and are not "representative". However, sometimes, they naturally belong to the dataset, and expert insight is needed.
- Fix formatting.

  For example, the Date may be expressed in many ways, and a simple Yes/No answer.
- Resolve missing values (by either removing the whole examples or imputing)

  Many methods have been developed for missing values imputation. It is a susceptible issue because new values may strongly bias the model.
- Remove duplicates.

The above steps often affect the training and need expertise in the application domain.

Later in this course, we will discuss techniques for data cleaning.

| ID | Age | Income | Gender | Customer_Satisfaction |
|---|---|---|---|---|
| 1 | 38 | 46641.356413713 | nan | Unsatisfied |
| 2 | 42 | 49129.0615585107 | female | Neutral |
| 3 | 18 | 119965.049731014 | Male | nan |
| 4 | 18 | 66828.0762224329 | nan | very unsatisfied |
| 5 | 58 | 57422.2721106762 | female | very unsatisfied |
| 6 | 28 | 59502.8174855665 | Other | Satisfied |
| 7 | 18 | 42659.6675768587 | Other | Neutral |
| 8 | 18 | 54019.1173206374 | Other | Satisfied |
| 9 | 40 | 25429.1604541137 | female | Unsatisfied |
| 10 | 21 | 15595.5862129548 | Other | Satisfied |
| 11 | 18 | 58094.2328460069 | Other | very unsatisfied |
| 12 | 18 | 39097.3278583155 | female | Very Satisfied |
| 13 | 30 |  | Other | Satisfied |
| 14 | 50 | 30617.3914472273 | Female | Very Satisfied |
| 15 | 18 |  | nan | Neutral |
| 16 | 34 | 39902.4430953214 | male | nan |
| 17 | 49 | 68381.6997683133 | Female | Very Satisfied |
| 18 | 33 | 44796.0962271524 | Other | Very Satisfied |
| 19 | 47 | 39218.9560738814 | Female | very unsatisfied |
| 20 |  | 14544.9226784447 | Other | Satisfied |

# Prepare Data

Unlike cleaning, which is application-dependent, data preparation/transformation is model-dependent. This usually subsumes:

- **Scaling**: Settings values of inputs to a similar range.

  Some models, especially those utilizing distance, are sensitive to large differences between input sizes.

- **Encoding**: Encode non-numeric data using real-valued vectors.

  Many models, especially those based on geometry, work only with numeric data. Non-numeric data such as Yes/No, Short/Medium/Long must be encoded appropriately.

- **Binning or Discretization** Convert continuous features into discrete bins to capture patterns in ranges.

**Comment:** Sometimes **Normalization**, that is changing the distribution of inputs to resemble the normal distribution, is mentioned. However, this step is typically not essential for machine learning itself. However, it is important to use statistical inference to test the significance of learned parameters.

# Prepare Data

- **Feature selection** Throw out input features that are too "similar" to other features.

  For example, if the temperature is measured both in Celsius and in Kelvin, keep one of them. The relationship can, of course, be a more complex (non-linear) correlation.

- **Dimensionality reduction** Transforming data from $\mathbb{R}^n$ to $\mathbb{R}^m$ where $m << n$.

  Growing dimension means growing difficulty of training for all models. Some models cease to work for high-dimensional data. The reduction typically searches for a few important characteristic features of inputs.

- **Feature aggregation** Introducing new features using operations on the original ones.

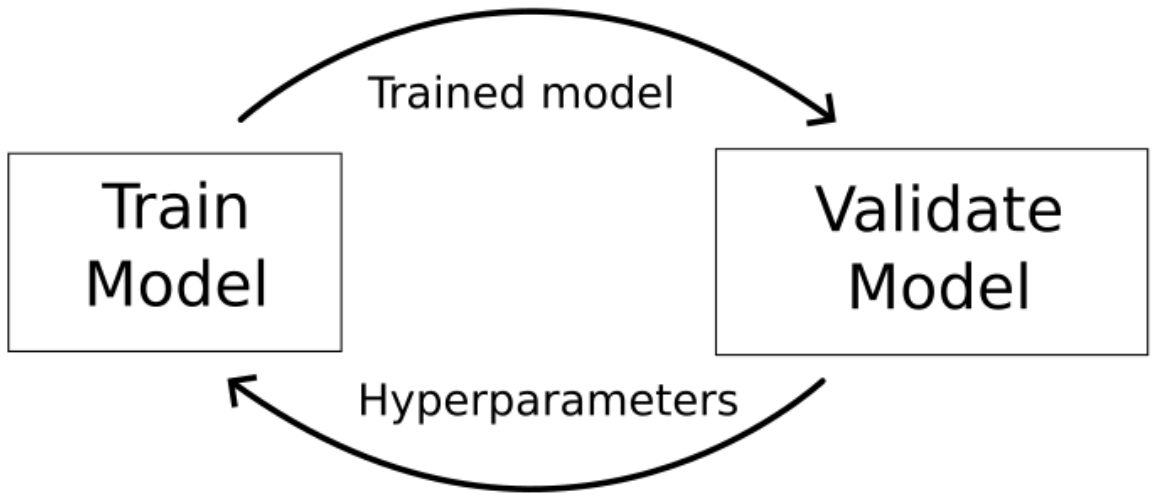
  We will see kernel transformations later in this course, allowing simple models to solve complex problems.

# Train Model

Now the dataset has been cleaned; we may train a model.

Before training, we should split the dataset into

- *training* dataset on which the model will learn
- *validation* dataset on which we fine-tune hyperparameters




The resulting model is obtained after several iterations of the above process.

# Evaluate Model

Here, we use the test set that we separated during data fetching.

In some cases, a brand new test set can be generated.

patients are examined regularly, creating new records continuously.

In some cases, it is tough to obtain new data.

For example, new expensive and difficult measurements are needed to obtain new data.

**Critical issue:** Make sure that you are truly testing

exactly the whole inference process.

Often, just a model is tested, and the testing and production inference engines are separated. This leads to truly nasty errors in the production!

We will discuss various generic metrics helpful in measuring the quality of the resulting model.

# Deploy to Production

Deployment of machine learning models is a complex question, application dependent.

The recently emerging area of MLOps is concerned with the engineering side of the model deployment.

From the technical point of view, the typical issues solved by ML Ops teams are

- how to extract/process data in real-time
- how much storage is required
- how to store/collect model (and data) artifacts/predictions
- how to set up APIs, tools, and software environments
- What the period of predictions (instantaneous or batch predictions) should be
- how to set up hardware requirements (or cloud requirements for on-cloud environments) by the computational resources required
- how to set up a pipeline for continuous training and parameter tuning

# Deploy to Production

From the user's point of view:

- How to get a sensible and valuable user output?
  - AI researchers will be satisfied with tons of running text in terminals.
  - "Normal" people need a graphical interface with understandable output.
  - Experts working in other domains typically demand speed and clarity at the extreme.
- How do you persuade users that the AI is working for them?
  - Especially if safety is at stake, you need to have outstanding arguments and explanations ready for end-users
  - In many areas, the devices need to be certified (medicine, automotive) for ML-based systems.

This complex subject will be only touched on in this course.

# Monitor, collect Data

Deployed machine learning models must be constantly monitored.

Because of the influx of new data, ML models work in highly dynamic environments.

For example, an image-processing medical diagnostic model suddenly misdiagnosed a patient because a nurse marked the sample with a marker pen.

Every customer has a different infrastructure and may produce data slightly differently.

Data for retraining and improvement should be stored.

Also, many areas allow the *active learning* where users provide feedback for (continuous) retraining of the models.

# Data

# Data Science Example

You receive data from a medical researcher concerning a project that you are eager to work on.

The data consists of a 1000 lines table with five columns:

| | | | | |
|---|---|---|---|---|
| 012 | 232 | 33.5 | 0 | 10.7 |
| 020 | 121 | 16.9 | 2 | 210.1 |
| 027 | 165 | 24.0 | 0 | 427.6 |
| | | ... | | |

The aim is to predict the last field given the others.

The medical researcher does not elaborate further on the data, but they seem to be pretty easy to work with, right?

After a few days, you have trained a model that predicts numbers resembling the ones in the table.

You contact the medical researcher and discuss the results.

# Model Discussion

**Researcher:** So, you got the data for all the patients?
**Data Miner:** Yes. I haven't had much time for analysis, but I do have a few interesting results.
**Researcher:** Amazing. There were so many data issues with this set of patients that I couldn't do much.
**Data Miner:** Oh? I didn't hear about any possible problems.
**Researcher:** Well, first, there is field 5, the variable we want to predict. It's common knowledge among people who analyze this type of data that results are better if you work with the log of the values, but I didn't discover this until later. Was it mentioned to you?
**Data Miner:** No.

# Model Dicsuccion

**Researcher:** But surely you heard about what happened to field 4? It's supposed to be measured on a scale from 1 to 10, with 0 indicating a missing value, but because of a data entry error, all 10's were changed into 0's. Unfortunately, since some of the patients have missing values for this field, it's impossible to say whether a 0 in this field is a real 0 or a 10. Quite a few of the records have that problem.
**Data Miner:** Interesting. Were there any other problems?
**Researcher:** Yes, fields 2 and 3 are basically the same, but I assume that you probably noticed that.
**Data Miner:** Yes, but these fields were only weak predictors of field 5.

# Model Discussion

**Researcher:** Anyway, given all those problems, I'm surprised you were able to accomplish anything.
**Data Miner:** True, but my results are really quite good. Field 1 is a very strong predictor of field 5. I'm surprised that this wasn't noticed before.
**Researcher:** What? Field 1 is just an identification number.
**Data Miner:** Nonetheless, my results speak for themselves.
**Researcher:** Oh, no! I just remembered. We assigned ID numbers after we sorted the records based on field 5. There is a strong connection, but it isn't very sensible. Sorry.

OK, what's the point?

You have to

Understand the task you want to solve and the data!

# Data Objects

*Data objects* represent entities we work with (e.g., classify them).

For example, in cancer prediction, the data objects are patients. In fruit classification, the data objects are individual fruits.

Data objects are described by *attributes* (or *features* or *variables*).

For example, the age, weight, genetic profile, and other patient characteristics. Or the width and height of a fruit.

# Attributes vs Features vs Variables

The name differs from field to field.

So, the following names are usually used as synonyms:

- *Attributes* - used mostly by database and data mining experts.
- *Features* - used mostly by machine learning experts.
- *Variables* - used mostly by statisticians.

One may make some distinctions

- *Attributes* represent information about the object without any additional assumptions.
- *Features* assume that their values are somewhat characteristic of the object.
- *Variables* assume that there is some process behind them (typically a random process in the case of statistics).

# Data Types - Categorical Attributes

*Categorical attributes* (nominal attributes) are symbols or names of things.

- Each value represents some kind of category, code, or state.
- Values are not ordered and should not be used quantitatively (in computer science, the values are known as enumerations).
- **Examples:**

$$\text{hair\_color} \in \{\text{black}, \text{brown}, \text{blond}, \text{red}, \text{auburn}, \text{gray}, \text{white}\}$$

$$\text{marital\_status} \in \{\text{single}, \text{married}, \text{divorced}, \text{widowed}\}$$

$$\text{customer\_ID} \in \{0, 1, 2, \ldots\}$$

  Even though the last one is usually expressed using numbers, it should not be used quantitatively.

*Binary attributes* are categorical attributes with only two values.

# DataTypes - Ordinal Attributes

*Ordinal attribute* is an attribute with values that have a meaningful order or ranking among them.

**Examples:**

$$\text{drink\_size} \in \{\text{small}, \text{medium}, \text{large}\}$$

$$\text{grades} \in \{\text{A}, \text{B}, \text{C}, \text{D}, \text{E}, \text{F}\}$$

It can also be obtained by discretizing numeric quantities into series of intervals.

Ordinal attributes do not allow arithmetic operations.

Categorical and ordinal attributes are called *qualitative* attributes.

Next, we look at numeric, i.e., *quantitative* attributes.

# Data Types - Numeric Attributes

*Numeric attributes* are quantities represented by numbers.

Distinguish two types: *Interval-scale* and *ratio-scale*.

| | **INTERVAL SCALE** | **RATIO SCALE** |
|---|---|---|
| Measurement interval | Equal intervals between consecutive points. | Equal intervals with the presence of a true zero. |
| Absolute zero | Lacks a true zero point. | Possesses a true zero point. |
| Statistical analysis | Limited to addition and subtraction | Allows for meaningful multiplication and division. |
| Meaningful ratios | Ratios are not meaningful due to the lack of zero. | Ratios are meaningful due to the presence of zero. |
| Examples | IQ scores, Celsius temperature, NPS data, etc. | Height, weight, income, etc. |

# Discrete vs Continuous Attributes

Often, two kinds of numeric attributes are distinguished:

- *Discrete*

  A finite or countably infinite range of values, i.e., integers may represent the values.

  Some (but not all) authors count the qualitative (categorical, ordinal) attributes among the discrete attributes.

- *Continuous*

  An uncountably infinite range of values, typically an interval.

  There are several more or less formal definitions of continuous attributes in the literature. For example:

  - All non-discrete variables.
  - Have an infinite number of values between any two values.
  - Their values are measured (??).

Deeper characteristics of data (statistical properties, etc.) will be examined at tutorials.

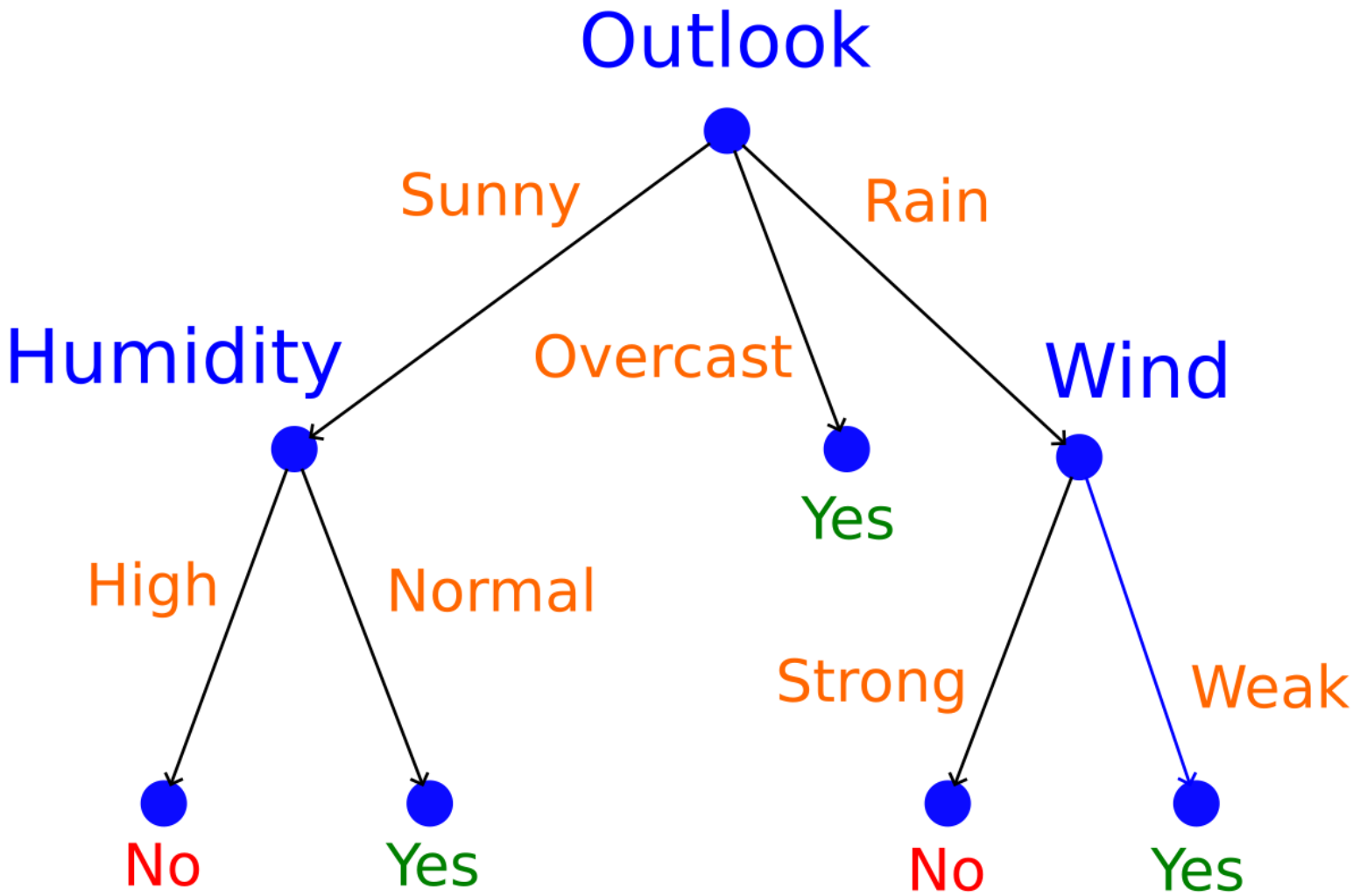
# Decision Trees

# Decision Trees

- One of the widely used methods for machine learning.
- Intuitively simple, directly explainable.
- Basis for random forests (a powerful model).

- We will consider the ID3 algorithm.
  Quinlan, 1979
- Various adjustments that appear in C4.5, CART, etc.

Consider the weather forecast for tennis playing. How would you decide whether to play today?




How do we obtain such a tree based on experience/data?

# Learning Decision Trees

Consider data represented as follows:

- A finite set of *attributes* $\mathcal{A} = \{A_1, \ldots, A_n\}$.
- Each attribute $A \in \mathcal{A}$ has its *set of values* $V(A)$.

We start with trees on discrete datasets, that is, assume $V(A)$ finite for all $A \in \mathcal{A}$.

Objects to be classified are described by vectors of values of all attributes:

$$\vec{x} = (x_1, \ldots, x_n) \in V(A_1) \times \cdots \times V(A_n)$$

Given $\vec{x}$ and an attribute $A_k$ we denote by $A_k(\vec{x})$ the value $x_k$ of the attribute $A_k$ in $\vec{x}$.

Consider a set $C$ of *classes*.

We consider a multiclass classification in general, i.e., $C$ is an arbitrary finite set.

# Example

The tennis problem:

- The attributes are:

$$A_1 = \textit{Outlook}, A_2 = \textit{Temperature}, A_3 = \textit{Humidity}, A_4 = \textit{Wind}$$

# Example

The tennis problem:

- The attributes are:

  $A_1 = \textit{Outlook}, A_2 = \textit{Temperature}, A_3 = \textit{Humidity}, A_4 = \textit{Wind}$

- The sets of values of the attributes:
  - $V(A_1) = \{\textit{Sunny}, \textit{Overcast}, \textit{Rain}\}$
  - $V(A_2) = \{\textit{Hot}, \textit{Mild}, \textit{Cool}\}$
  - $V(A_3) = \{\textit{High}, \textit{Normal}\}$
  - $V(A_4) = \{\textit{Strong}, \textit{Weak}\}$

# Example

The tennis problem:

- The attributes are:

$$A_1 = \textit{Outlook}, A_2 = \textit{Temperature}, A_3 = \textit{Humidity}, A_4 = \textit{Wind}$$

- The sets of values of the attributes:
  - $V(A_1) = \{\textit{Sunny}, \textit{Overcast}, \textit{Rain}\}$
  - $V(A_2) = \{\textit{Hot}, \textit{Mild}, \textit{Cool}\}$
  - $V(A_3) = \{\textit{High}, \textit{Normal}\}$
  - $V(A_4) = \{\textit{Strong}, \textit{Weak}\}$
- Consider

$$\vec{x} = (\textit{Overcast}, \textit{Hot}, \textit{Normal}, \textit{Weak}) \in V(A_1) \times V(A_2) \times V(A_3) \times V(A_4)$$

# Example

The tennis problem:

- The attributes are:

$$A_1 = \textit{Outlook}, A_2 = \textit{Temperature}, A_3 = \textit{Humidity}, A_4 = \textit{Wind}$$

- The sets of values of the attributes:
  - $V(A_1) = \{\textit{Sunny}, \textit{Overcast}, \textit{Rain}\}$
  - $V(A_2) = \{\textit{Hot}, \textit{Mild}, \textit{Cool}\}$
  - $V(A_3) = \{\textit{High}, \textit{Normal}\}$
  - $V(A_4) = \{\textit{Strong}, \textit{Weak}\}$
- Consider

$$\vec{x} = (\textit{Overcast}, \textit{Hot}, \textit{Normal}, \textit{Weak}) \in V(A_1) \times V(A_2) \times V(A_3) \times V(A_4)$$

Then

$$A_3(\vec{x}) = \textit{Humidity}(\vec{x}) = \textit{Normal}$$

$$A_4(\vec{x}) = \textit{Wind}(\vec{x}) = \textit{Weak}$$

# Example

The tennis problem:

- The attributes are:

  $A_1 = \mathit{Outlook}, A_2 = \mathit{Temperature}, A_3 = \mathit{Humidity}, A_4 = \mathit{Wind}$

- The sets of values of the attributes:
  - $V(A_1) = \{\mathit{Sunny}, \mathit{Overcast}, \mathit{Rain}\}$
  - $V(A_2) = \{\mathit{Hot}, \mathit{Mild}, \mathit{Cool}\}$
  - $V(A_3) = \{\mathit{High}, \mathit{Normal}\}$
  - $V(A_4) = \{\mathit{Strong}, \mathit{Weak}\}$
- Consider

  $$\vec{x} = (\mathit{Overcast}, \mathit{Hot}, \mathit{Normal}, \mathit{Weak}) \in V(A_1) \times V(A_2) \times V(A_3) \times V(A_4)$$

  Then

  $$A_3(\vec{x}) = \mathit{Humidity}(\vec{x}) = \mathit{Normal}$$

  $$A_4(\vec{x}) = \mathit{Wind}(\vec{x}) = \mathit{Weak}$$

- $C = \{\mathit{Yes}, \mathit{No}\}$

# Decision Trees

Consider (directed, rooted) *trees* $\mathcal{T} = (T, E)$ where $T$ is a set of nodes and $E \subseteq T \times T$ is a set of directed edges.

Denote by $T_{leaf} \subseteq T$ the set of all *leaves* of the tree and by $T_{int}$ the set $T \setminus T_{leaf}$ of *internal nodes*.

A *decision tree* is

- a tree $\mathcal{T} = (T, E)$ where
- each leaf $\tau \in T_{leaf}$ is assigned a class $C(\tau) \in C$,
- each internal node $\tau \in T_{int}$ is assigned an attribute $A(\tau) \in \mathcal{A}$,
- and there is a bijection between edges from $\tau$ and values of the attribute $A(\tau)$. Given an edge $(\tau, \tau') \in E$ we write $V(\tau, \tau')$ to denote the value of the attribute $A(\tau)$ assigned to the edge.

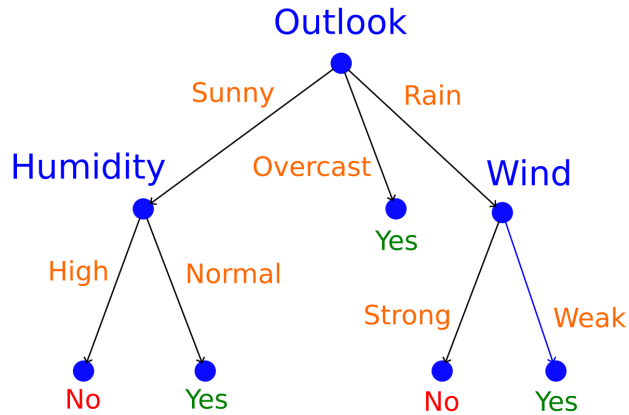
**Inference:** Given an input $\vec{x}$, we traverse the tree from the root to a leaf, always choosing edges labeled with values of attributes from $\vec{x}$. The output is the class labeling the leaf.

# Example

$$T = \{O, H, W, z_1, z_2, z_3, z_4, z_5\}$$

$$T_{leaf} = \{z_1, z_2, z_3, z_4, z_5\},\ T_{int} = \{O, H, W\}$$

$$E = \{(O, H), (O, W), (H, z_1), (H, z_2), (O, z_3), (W, z_4), (W, z_5)\}$$




$$C(z_1) = C(z_3) = \textit{No},\ C(z_2) = C(z_4) = \textit{Yes}$$

$$A(O) = \textit{Outlook},\ A(H) = \textit{Humidity},\ A(W) = \textit{Wind}$$

$$V(O, H) = \textit{Sunny},\ V(O, z_3) = \textit{Overcast},\ V(O, W) = \textit{Rain}$$
$$V(H, z_1) = \textit{High},\ V(H, z_2) = \textit{Normal}$$
$$V(W, z_4) = \textit{Strong},\ V(W, z_5) = \textit{Weak}$$

**Inference:** For $(\textit{Rain}, \textit{Hot}, \textit{High}, \textit{Strong})$ we reach $z_4$, yielding $\textit{No}$.

# Training Dataset

Consider a *training dataset*

$$\mathcal{D} = \{(\vec{x}_k, c_k) \mid k = 1, \ldots, p\}$$

Here $\vec{x}_k \in V(A_1) \times \cdots \times V(A_k)$ and $c_k \in C$ for every $k$.

Technically $\mathcal{D}$ can be a multiset containing several occurrences of the same vector.

| Index | Outlook | Temperature | Humidity | Wind | PlayTennis |
|---|---|---|---|---|---|
| 1 | Sunny | Hot | High | Weak | No |
| 2 | Sunny | Hot | High | Strong | No |
| 3 | Overcast | Hot | High | Weak | Yes |
| 4 | Rain | Mild | High | Weak | Yes |
| 5 | Rain | Cool | Normal | Weak | Yes |
| 6 | Rain | Cool | Normal | Strong | No |
| 7 | Overcast | Cool | Normal | Strong | Yes |
| 8 | Sunny | Mild | High | Weak | No |
| 9 | Sunny | Cool | Normal | Weak | Yes |
| 10 | Rain | Mild | Normal | Weak | Yes |
| 11 | Sunny | Mild | Normal | Strong | Yes |
| 12 | Overcast | Mild | High | Strong | Yes |
| 13 | Overcast | Hot | Normal | Weak | Yes |
| 14 | Rain | Mild | High | Strong | No |

$$\mathcal{D} = \{((Sunny, Hot, High, Weak), No),$$
$$((Sunny, Hot, High, Strong), No)$$
$$\ldots$$
$$((Rain, Mild, High, Strong), No)\}$$

# Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.
- If there is just a single class in $\mathcal{D}$, create a single node decision tree that returns the class.
- Otherwise, identify an attribute $A \in \mathcal{A}$ which *best classifies* the examples in $\mathcal{D}$. For every $v \in V(A)$ we obtain

$$\mathcal{D}_v = \{\vec{x} \mid \vec{x} \in \mathcal{D}, A(\vec{x}) = v\}$$

  We aim to have each $\mathcal{D}_v$ as pure as possible, that is, ideally, to contain examples of just a single class.
- Finally,
  - create a root node $\tau$ of a decision tree,
  - assign the attribute $A$ to $\tau$,
  - for every $v \in V(A)$, recursively construct a decision tree with a root $\tau_v$ using $\mathcal{D}_v$,
  - for every $v \in V(A)$ introduce an edge $(\tau, \tau_v)$ assigned $v$.

1: **function** ID3(dataset $\mathcal{D}$, attribute set $\mathcal{A}$)
2: Create a root node $\tau$ for the tree
3: **if** $\mathcal{D} = \emptyset$ **then**
4: Return the single node $\tau$ assigned with a default class.
5: **else if** all examples in $\mathcal{D}$ are of the same class $c$ **then**
6: Return the single-node tree, where $\tau$ is assigned $c$
7: **else if** set of attributes $\mathcal{A}$ is empty **then**
8: Return the single-node tree where $\tau$ is assigned
the most common class in $\mathcal{D}$
9: **else**
10: Choose attribute $A \in \mathcal{A}$ *best classifying* examples in $\mathcal{D}$.
11: Set the decision attribute for $\tau$ to $A$
12: **for** each value $v \in D(A)$ **do**
13: Compute a decision tree ID3($\mathcal{D}_v, \mathcal{A} \setminus \{A\}$) with root $\tau_v$,
14: add a new edge $(\tau, \tau_v)$ assigned $v$.
15: **end for**
16: **end if**
17: **return** $\tau$
18: **end function**

# Best Classifying Attribute

We aim to choose an attribute that best informs us about the class.

As a result, we would possibly use as few attributes as possible and obtain a small tree containing only class-relevant decisions.

How to choose an attribute that best classifies examples in $\mathcal{D}$?

There are several measures used in practice.

The most common are

- *information gain*
- *Gini impurity decrease*

# Information Gain

The information gain is based on the notion of entropy.

We need some notation:

- Given a training dataset $\mathcal{D}$ and a class $c \in C$ we denote by $p_c$ the proportion of examples with class $c$ in $\mathcal{D}$.
- We define the *entropy* of $\mathcal{D}$ by

$$Entropy(\mathcal{D}) = \sum_{c \in C} -p_c \log_2 p_c$$

- The *information gain* of an attribute $A$ is then defined by

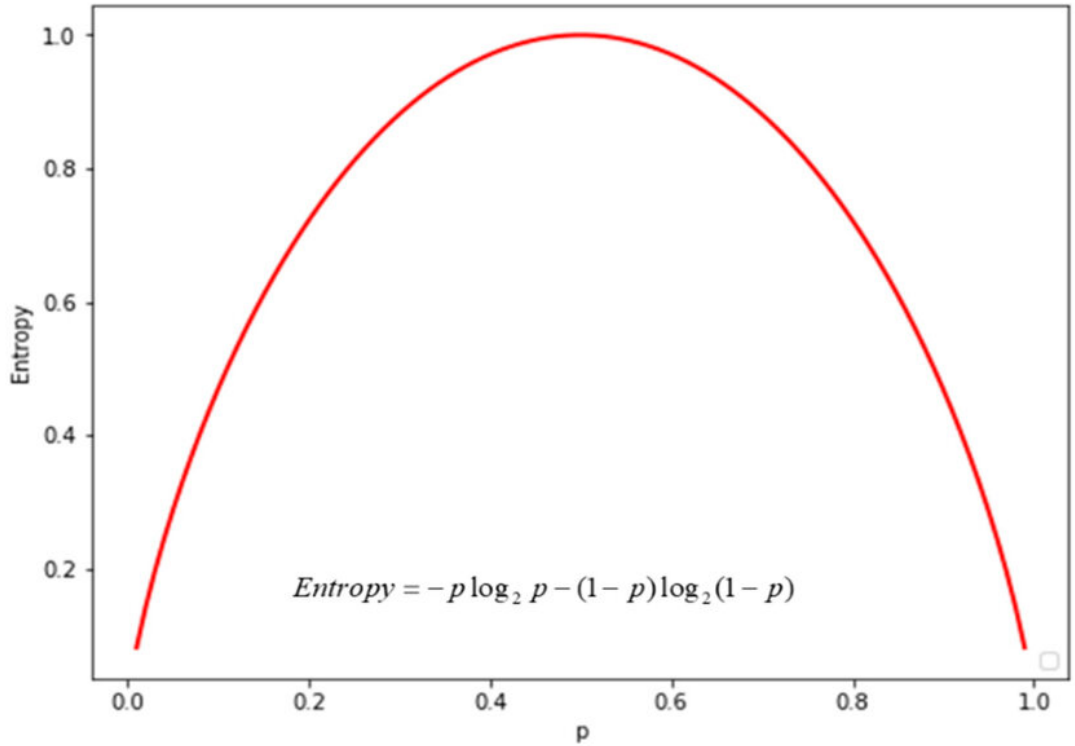
$$Gain(\mathcal{D}, A) = Entropy(\mathcal{D}) - \sum_{v \in V(A)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} Entropy(\mathcal{D}_v)$$

In every step of the ID3 algorithm, we choose an attribute *maximizing* the information gain for the current dataset $\mathcal{D}$.

# Information Gain

The intuition behind information gain:

- Consider $C = \{0, 1\}$ and $p$ the proportion of examples of class 1. $p$ measures the "uncertainty" of the class:




- $\sum_{v \in V(A)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} Entropy(\mathcal{D}_v)$ is weighted uncertainty of classes in each $\mathcal{D}_v$ (weighted by the relative size of $\mathcal{D}_v$).
- $Gain(\mathcal{D}, A)$ measures reduction in uncertainty of classes by splitting $\mathcal{D}$ according to $A$.

# Gini Impurity

- We define *Gini impurity* of $\mathcal{D}$ by

$$Gini(\mathcal{D}) = 1 - \sum_{c \in C} p_c^2$$

- The *impurity decrease* of an attribute $A$ is then defined similarly to the gain in the entropy case

$$ImpDec(\mathcal{D}, A) = Gini(\mathcal{D}) - \sum_{v \in V(A)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} Gini(\mathcal{D}_v)$$

In every step of the ID3 algorithm, we choose an attribute *maximizing* the impurity decrease for the current dataset $\mathcal{D}$.

# Gini Impurity

What is the intuition behind $Gini(\mathcal{D})$ ?

Assume we randomly independently choose objects from $\mathcal{D}$.

$1 - \sum_{c \in C} p_c^2$ is the probability of choosing two objects of different classes in two consecutive independent trials.

Indeed, $p_c$ is the probability of choosing an object of class $c$, $p_c^2$ the probability of choosing objects of the class $c$ twice, and $\sum_{c \in C} p_c^2$ the probability of choosing two objects of the same class.

In what follows (and at the exam), we will work only with the Gini impurity as it is easier to compute.

# Example

Consider our tennis example (see the table).

- Consider the whole dataset $\mathcal{D}$.
  - $p_{Yes} = 9/14$
  - $p_{No} = 5/14$
  - $Gini(\mathcal{D}) = 1 - (9/14)^2 - (5/14)^2 = 0.45918$
- For $A = Outlook$ we get
  - $Gini(\mathcal{D}_{Sunny}) = 1 - (2/5)^2 - (3/5)^2 = 0.48$
  - $Gini(\mathcal{D}_{Overcast}) = 1 - 1^2 - 0^2 = 0$
  - $Gini(\mathcal{D}_{Rain}) = 1 - (3/5)^2 - (2/5)^2 = 0.48$

  Thus

$$ImpDec(\mathcal{D}, Outlook) = 0.459 - (5/14) \cdot 0.48 - (4/14) \cdot 0 - (5/14) \cdot 0.48 = 0.117$$

- $ImpDec(\mathcal{D}, Temperature) = 0.018$
- $ImpDec(\mathcal{D}, Humidity) = 0.091$
- $ImpDec(\mathcal{D}, Wind) = 0.030$

So the largest information gain is given by the *Outlook*.

# Example

Going further on, consider $\mathcal{D} = \mathcal{D}_{Sunny}$. We get

- $ImpDec(\mathcal{D}, Temperature) = 0.279$
- $ImpDec(\mathcal{D}, Humidity) = 0.48$
- $ImpDec(\mathcal{D}, Wind) = 0.013$

The best choice attribude after *Sunny* in *Outlook* is *Humidity*.

Now consider $\mathcal{D} = \mathcal{D}_{Rain}$.

- $ImpDec(\mathcal{D}, Temperature) = 0.013$
- $ImpDec(\mathcal{D}, Humidity) = 0.013$
- $ImpDec(\mathcal{D}, Wind) = 0.48$

The best choice attribude after *Rain* in *Outlook* is *Wind*.

# Continuous-Valued Attributes

What if values of an attribute $A$ come from a continuous variable?

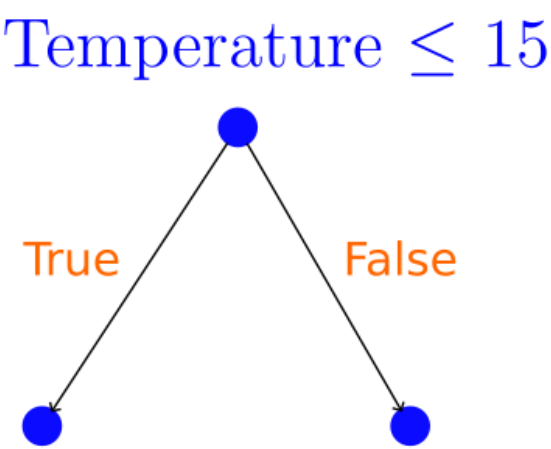
$A$ is a numerical attribute that can take any value in an interval, such as temperature, size, time, etc.

Consider an internal node $\tau \in T_{int}$ assigned such a continuous attribute $A$. Then

- $\tau$ is assigned a threshold value called a *cut point* $H \in \mathbb{R}$,
- there are two edges $e_{\text{true}}, e_{\text{false}}$ from $\tau$,
- $e_{\text{true}}$ labeled with True and $e_{\text{false}}$ labeled with False.

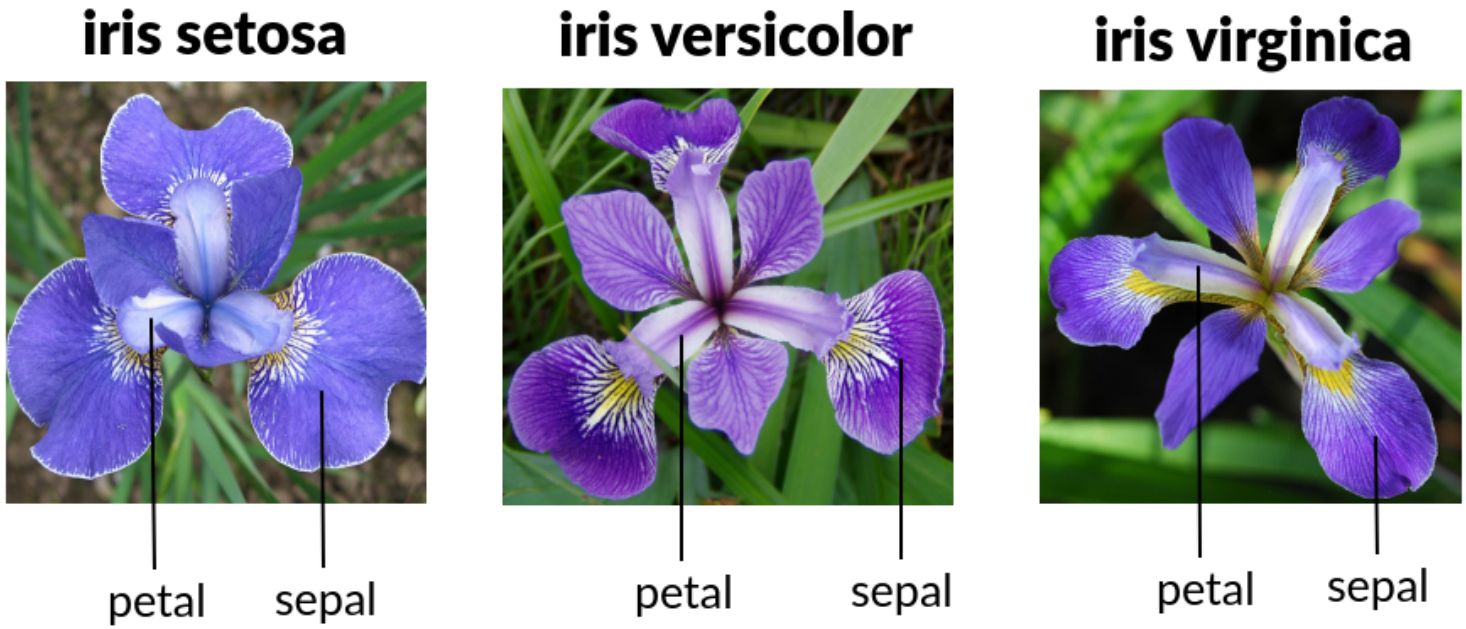
During inference, when considering an example $\vec{x}$ in the node $\tau$,

- evaluate $A(\vec{x}) \leq H$,
- if $A(\vec{x}) \leq H$, then follow $e_{\text{true}}$,
- else follow $e_{\text{false}}$.




In training, the cut point is chosen from the attribute values in the training set using information gain/impurity decrease similar to discrete attributes.

# Iris Example




**Attributes**

Sepal.Length, Sepal.Width, Petal.Length, Petal.Width

**Classes** (Variety)

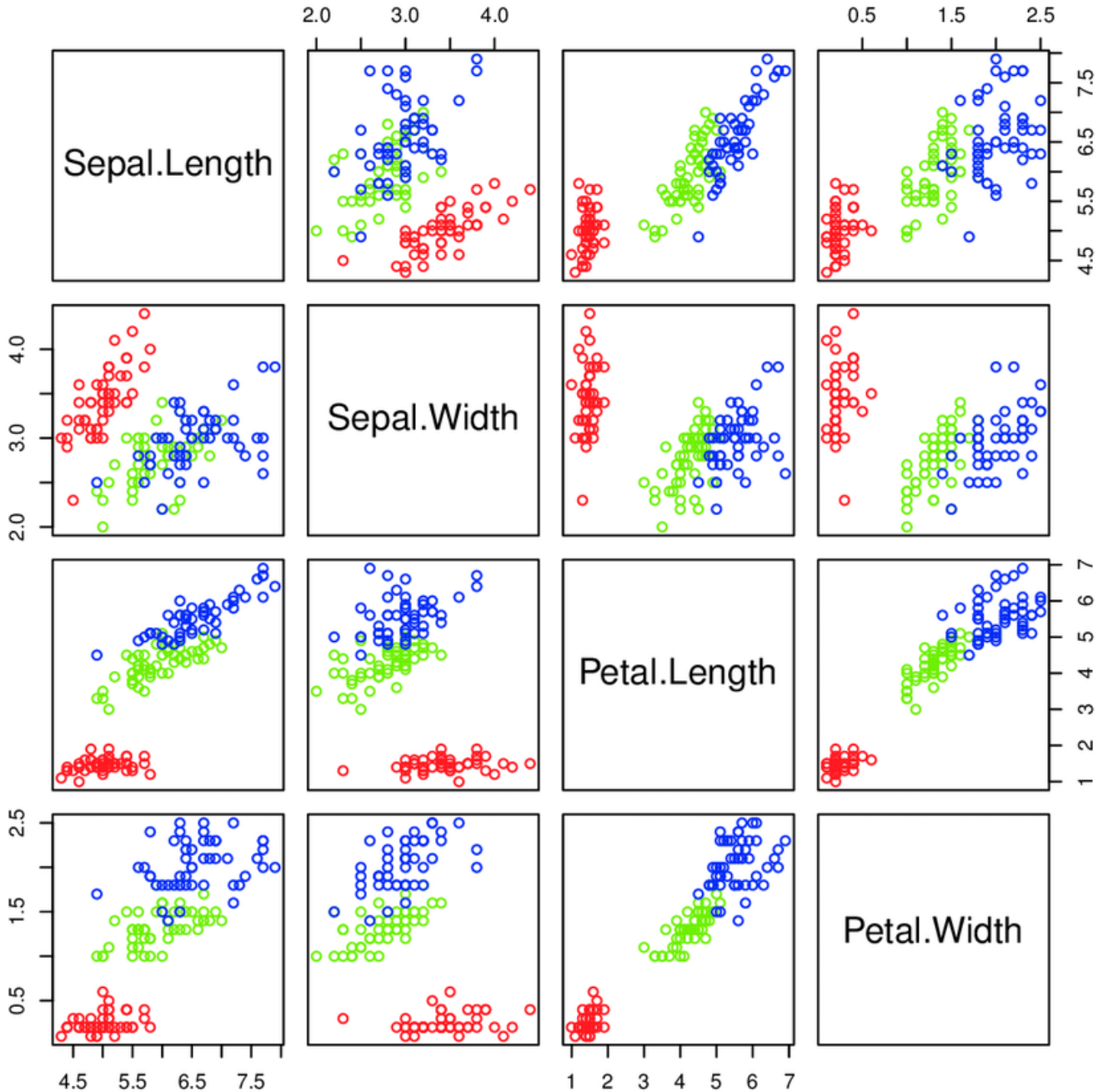
Setosa, Versicolor, Virginica

# Iris Example

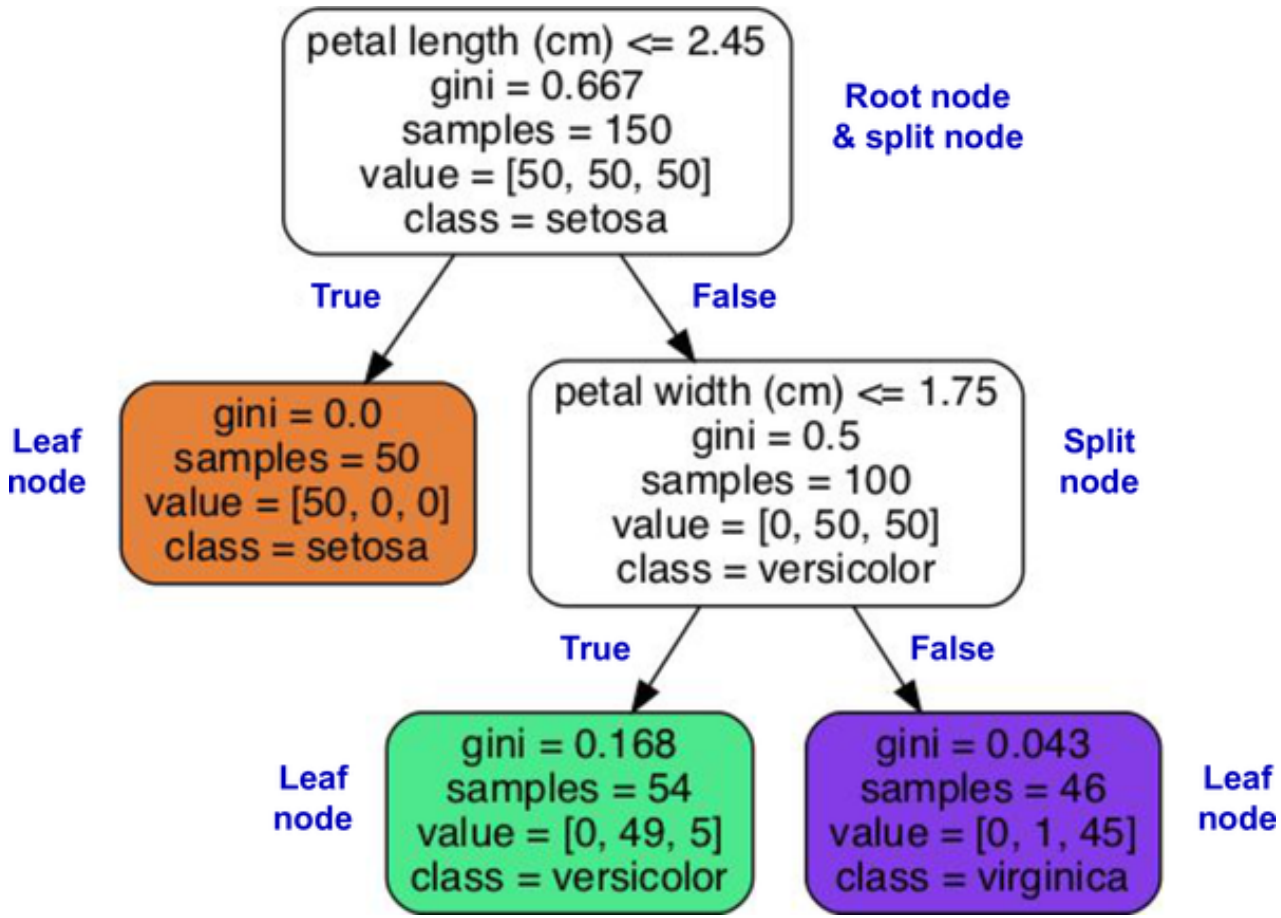
The dataset (150 examples):

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Variety |
|---|---|---|---|---|
| 5.5 | 3.5 | 1.3 | 0.2 | Setosa |
| 6.8 | 2.8 | 4.8 | 1.4 | Versicolor |
| 6.7 | 3.1 | 4.7 | 1.5 | Versicolor |
| 6.9 | 3.1 | 5.1 | 2.3 | Virginica |
| 7.3 | 2.9 | 6.3 | 1.8 | Virginica |
| 5.4 | 3.7 | 1.5 | 0.2 | Setosa |
| 4.6 | 3.4 | 1.4 | 0.3 | Setosa |
| 6.2 | 2.8 | 4.8 | 1.8 | Virginica |
| 5.4 | 3.0 | 4.5 | 1.5 | Versicolor |
| 4.7 | 3.2 | 1.6 | 0.2 | Setosa |
| 6.7 | 3.3 | 5.7 | 2.1 | Virginica |
| 5.0 | 3.4 | 1.5 | 0.2 | Setosa |
| 5.0 | 3.0 | 1.6 | 0.2 | Setosa |
| 4.4 | 2.9 | 1.4 | 0.2 | Setosa |
| 6.0 | 3.4 | 4.5 | 1.6 | Versicolor |
| 5.1 | 3.5 | 1.4 | 0.2 | Setosa |
| 6.6 | 3.0 | 4.4 | 1.4 | Versicolor |
| 5.9 | 3.2 | 4.8 | 1.8 | Versicolor |
| 5.6 | 2.8 | 4.9 | 2.0 | Virginica |
| ... | | | | |

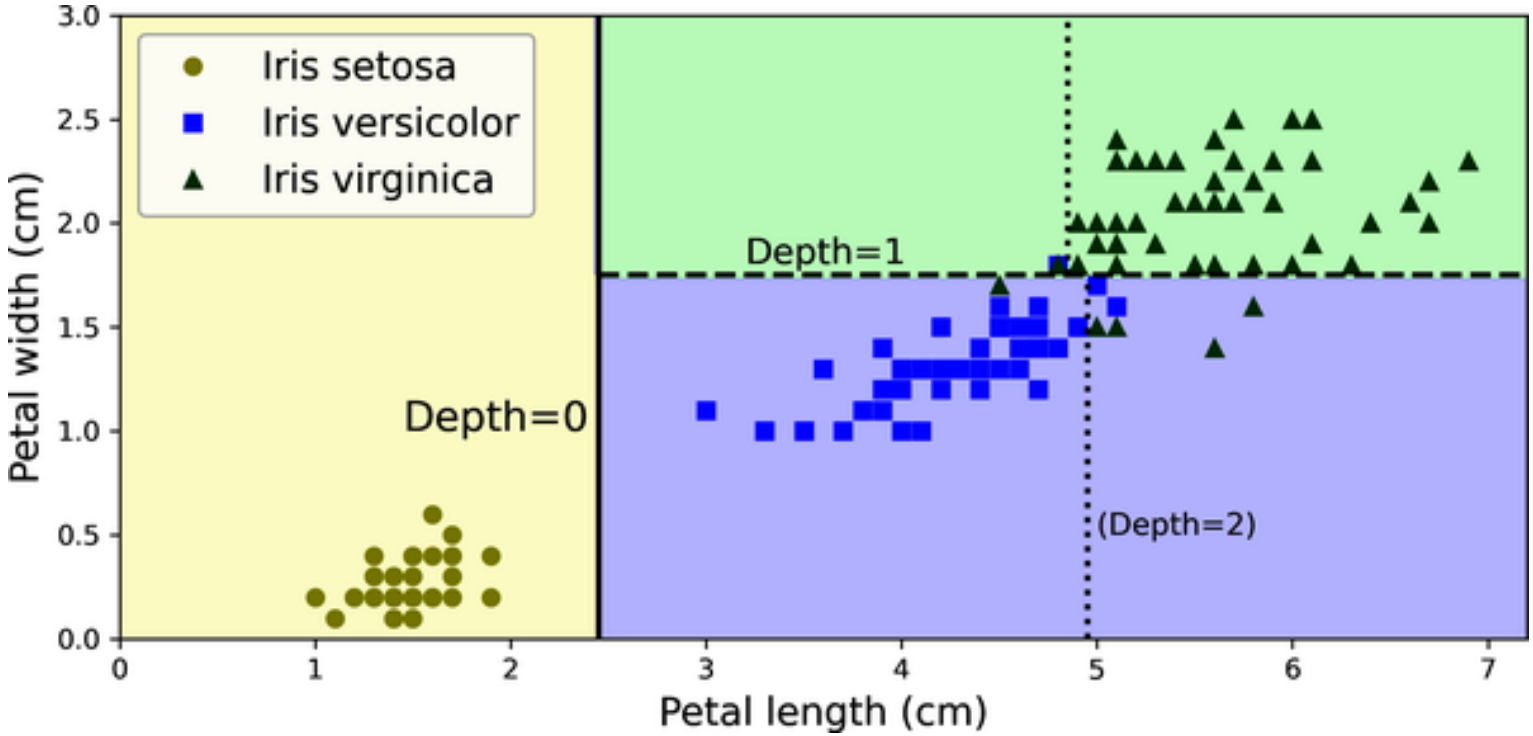
# Iris Example



# Iris Example - Decision Tree



# Iris Example - Decision Tree Boudaries




If the leaves are split further, the Depth $= 2$ boundary would be added.

# Attribute Importance Computation

How important are attributes for the trained tree $\mathcal{T}$? Depends on

- how close they are to the root of $\mathcal{T}$,
- how large information gain/decrease in impurity they give.

There are several formulae for computing the importance.

One example is *mean decrease impurity* defined as follows:

Consider a decision tree $T$ trained on a dataset $\mathcal{D}$ using the ID3.

For every node $\tau$ of $\mathcal{T}$, denote by $\mathcal{D}[\tau]$ the subset of $\mathcal{D}$ which was used in the ID3 procedure when the node $\tau$ was created (line 2).

Consider an attribute $A$ and denote by $T[A] \subseteq T_{int}$ the set of all nodes of $\mathcal{T}$ assigned the attribute $A$ by ID3 (line 11).

Then define the importance as the average decrease in Gini impurity (i.e., average *ImpDec*) in the nodes of $T[A]$:

$$GiniImportance(A) = \sum_{\tau \in T[A]} \frac{|\mathcal{D}[\tau]|}{|\mathcal{D}|} ImpDec(\mathcal{D}[\tau], A)$$

# Decision Trees

Practical Issues

# Practical Issues

- Data preprocessing
- Model tunning (overfitting and underfitting)
- Sensitivity to changes in data/hyperparameters
- Learning representation problems (the XOR)

# Data Preprocessing

Little preprocessing is needed for decision trees.

Of course, ideally, clean up the data, but

- Missing values are not such a big issue
  (considering a concrete example, exclude the attributes with missing values)
- Outliers are not such a big issue either; the division of nodes is done based on relative counts, not concrete values.
- Decision trees can cope with continuous and categorical values directly (i.e., no encoding necessary)

Imbalanced classes might cause problems because of small information gain/impurity decrease in splitting.

# Imbalanced Classes

Consider a dataset $\mathcal{D}$ where

- there are two classes, $C = \{0, 1\}$,
- $10^6$ examples have the class 1,
- 100 examples have the class 0.

Then

- $p_1 = 10^6/(10^6 + 100) \approx 1$ and $p_0 = 100/10^6 \approx 0$,
- thus the Gini impurity $1 - p_1^2 - p_0^2 \approx 0$.

Consider an attribute $A$ with $V(A) = \{a, b\}$.

Splitting $\mathcal{D}$ according to $A$ gives to sets $\mathcal{D}_a$ and $\mathcal{D}_b$.

What is the impurity decrease caused by the attribute?

$$ImpDec(\mathcal{D}, A) = Gini(\mathcal{D}) - \frac{|\mathcal{D}_a|}{|\mathcal{D}|} Gini(\mathcal{D}_a) - \frac{|\mathcal{D}_b|}{|\mathcal{D}|} Gini(\mathcal{D}_b)$$

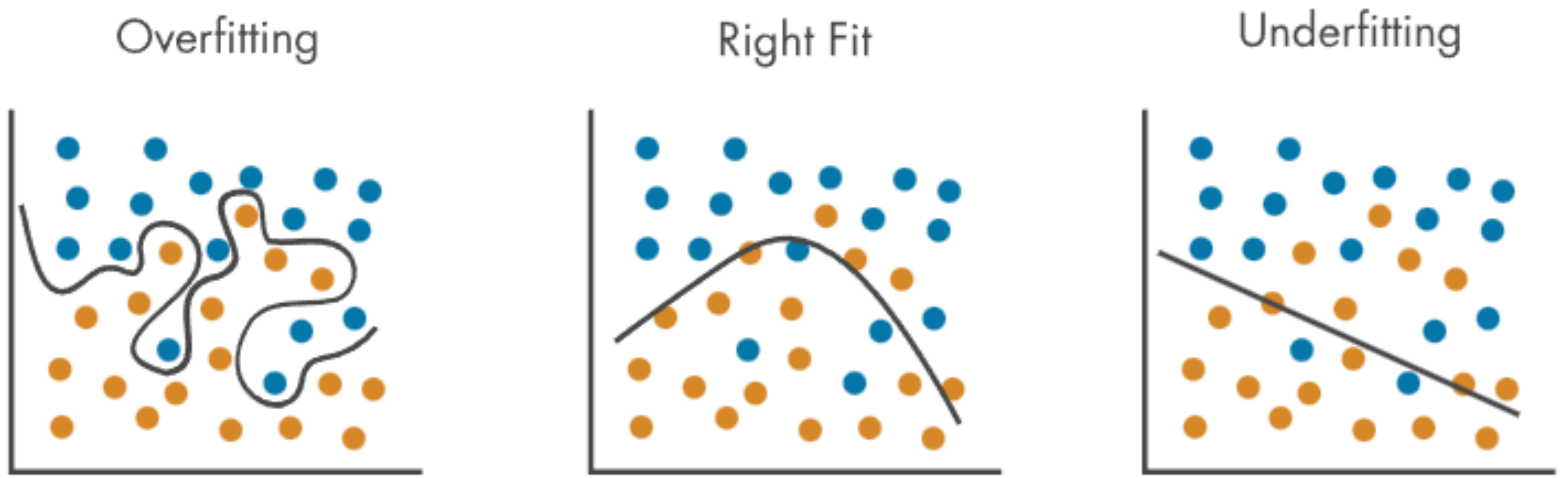
For small $|\mathcal{D}_a|$ (say $\leq 1000$) we have small $|\mathcal{D}_a|/|\mathcal{D}|$

For not so small $\mathcal{D}_a$ we have $Gini(\mathcal{D}_a) \approx 0$.

In both cases, the impurity decrease is very small.

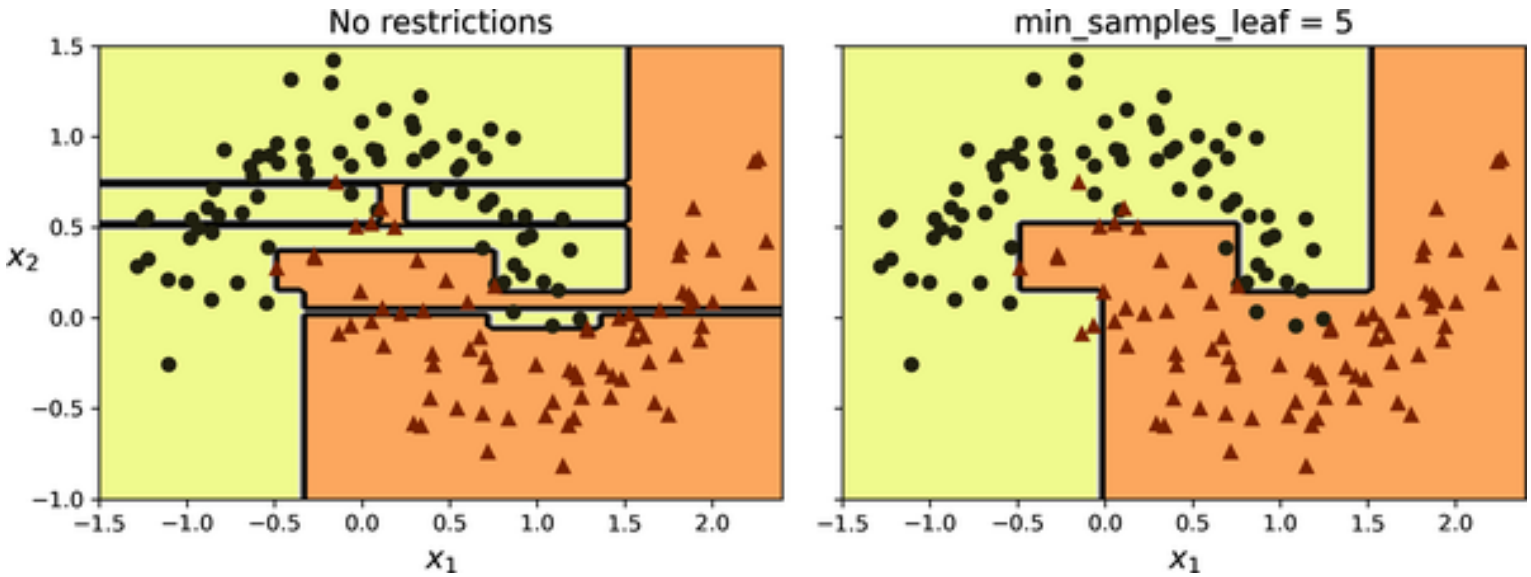
# Model Tuning - Over/Under Fitting

The behavior of the model on the training set:




- The left one is strongly overfitting. It would possibly not work well on new data.
- The right one is strongly underfitting. It would probably give poor classification results.
- The middle one seems good (but still needs to be tested on fresh data).

# Model Tuning - Overfitting in Decision Trees

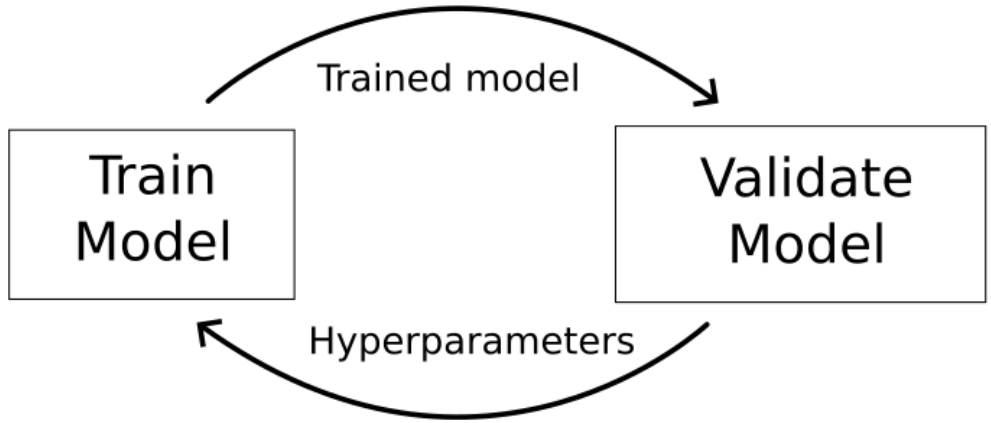



See the overfitting on the left and the "nice" model on the right.

Both overfitting and underfitting are best avoided. But how do we find out?

# Model Tuning (In General)

Recall from the first lecture:




The validation should be done on a **validation set** separated from the training set.

We will discuss more sophisticated techniques later.

What hyperparameters to set? (see the next slide)

What to observe? In the case of decision trees, one should observe the difference between performance measures (e.g., classification accuracy) on the training and validation sets.

The too-large difference implies an improperly fitting model.

# How to Fit Decision Trees?

There are several approaches available for decision trees.

Generally, the overfitting can be either prevented or resolved.

- Pre-pruning: Build the tree so it does not overfit by restricting its size.
- Post-pruning: Overfit with a large tree and remove subtrees to make it smaller.
- Ensemble methods: Fit several different trees and let them classify together (e.g., using majority voting).

The post-pruning approach has been more successful in practice than the pre-pruning because it is usually hard to say when to stop growing the tree.

We shall meet this controversy also in deep learning, where recent history shows a similar phenomenon.

The ensemble methods will be covered later when we discuss random forests.

# Pre-Prunning - Hyperparamaters

Hyperparameters controlling the size of the tree:

- Maximum depth - do not grow the tree beyond the max depth

  The deeper the tree, the more complex models you can create $\Rightarrow$ overfitting. Low depth may restrict expressivity.

- Minimum number of examples to split a node - if $\mathcal{D}[\tau]$ is small, $\tau$ becomes a leaf (labeled with the majority class)

  Higher values prevent a model from learning relations specific only for a few examples. Too high values may result in underfitting.

- Minimum number of examples required to be in a leaf

  Similar to the previous one. A higher number means we cannot have very specific branches concerned with particular combinations of values.

- Minimum information gain/impurity decrease

  A small impurity decrease means that the split does not contribute too much to the classification (their proportions after a split are similar to proportions before a split). However, keep in mind that it is *weighted average impurity* after the split.

# Post-Pruning - Reduced Error Pruning

Train a large tree and then remove nodes that make classification worse on the validation set.

Given a decision tree $\mathcal{T}$ and its internal node $\tau \in \mathcal{T}_{int}$, we denote by $\mathcal{T}_{-\tau}$ the tree obtained from $\mathcal{T}$ by removing the subtree rooted in $\tau$, i.e., $\tau$ is a leaf of $\mathcal{T}_{-\tau}$.

1: Train $\mathcal{T}$ to maximum fit on the *training dataset.*
2: **while** true **do**
3: $\quad Err[\mathcal{T}] \leftarrow$ the error of $\mathcal{T}$ on the validation set.
4: $\quad$ **for** $\tau \in \mathcal{T}_{int}$ **do**
5: $\quad\quad Err[\mathcal{T}_{-\tau}] \leftarrow$ the error of $\mathcal{T}_{-\tau}$ on the validation set.
6: $\quad$ **end for**
7: $\quad$ **if** $Err[\mathcal{T}] \leq \min\{Err[\mathcal{T}_{-\tau}] \mid \tau \in \mathcal{T}_{int}\}$ **then return** $\mathcal{T}$
8: $\quad$ **else**
9: $\quad\quad \mathcal{T} \leftarrow argmin\{Err[\mathcal{T}_{-\tau}] \mid \tau \in \mathcal{T}_{int}\}$
10: $\quad$ **end if**
11: **end while**

The error $Err[\mathcal{T}]$ can be any measure of the "badness" of the decision tree $\mathcal{T}$. For example, $1 - Accuracy$.

# Other Pruning Methods

There are more pruning methods.

- Rule Post-Pruning:
  - Transform the tree into a set of rules.
    Rules correspond to paths in the tree; they have a form of implication: Specific values of attributes imply a class.
  - Remove the attribute conditions from the premises of the implications.

  This gives a more refined pruning: Instead of removing the whole subtree, each path of the subtree can be pruned individually.
- Using cost complexity measure: Evaluate trees not only based on the classification error but also based on their size.

  Typically introduce regularization into the error functions: Given a decision tree $\mathcal{T}$

  $$Err_\alpha(\mathcal{T}) = Err(\mathcal{T}) + \alpha|\mathcal{T}|$$

  The original paper by Breiman et al. (1984) defined $Err(\mathcal{T})$ to be the misclassification rate on the training dataset, and $|\mathcal{T}|$ is the number of nodes of the tree $\mathcal{T}$.

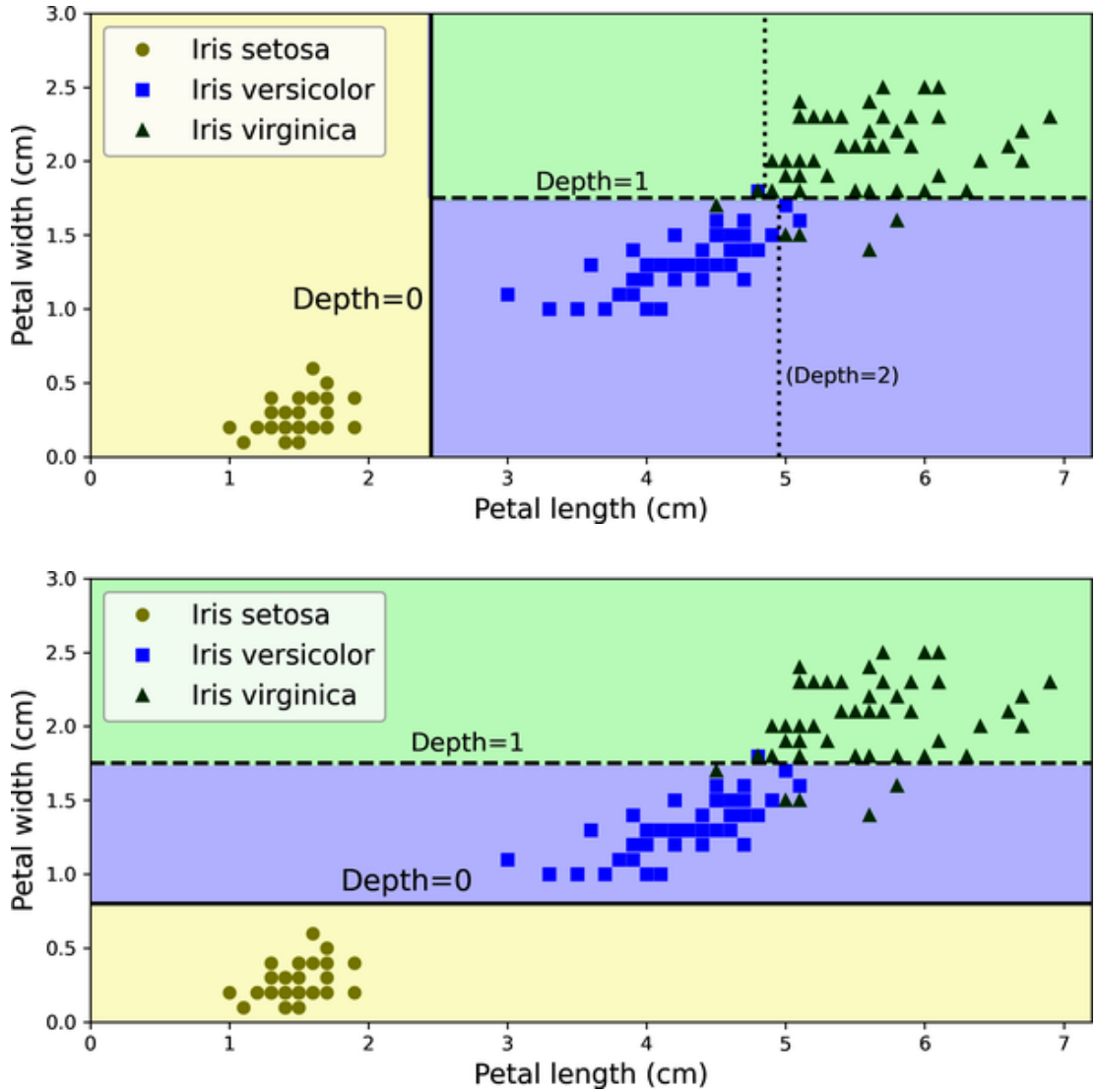
# Sensitivity to Small Changes and Randomness

- Decision trees are sensitive to small changes in data and hyperparameters.
  Several attributes may provide (almost) identical information gain but divide the training dataset very differently.
- Some implementations choose attributes partially in random (sci-kit-learn). You may get completely different trees.

A solution is to train an ensemble of many decision trees and then use majority voting for classification.

This is the fundamental idea behind random forests (see later lectures).

# Iris - Illustration

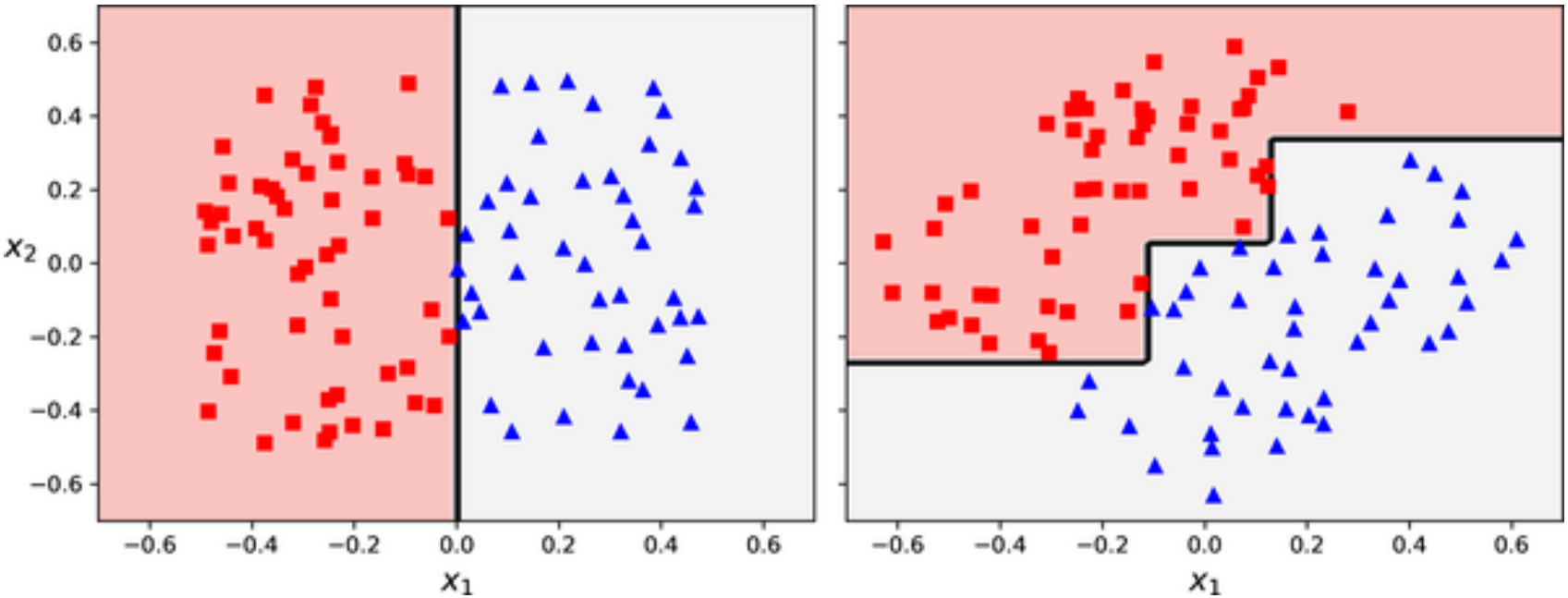



Decision trees trained on the Iris dataset.

Iris Setosa is perfectly separated by many choices for the first split.

# Axis Sensitivity

The decision makes divisions along particular axes:

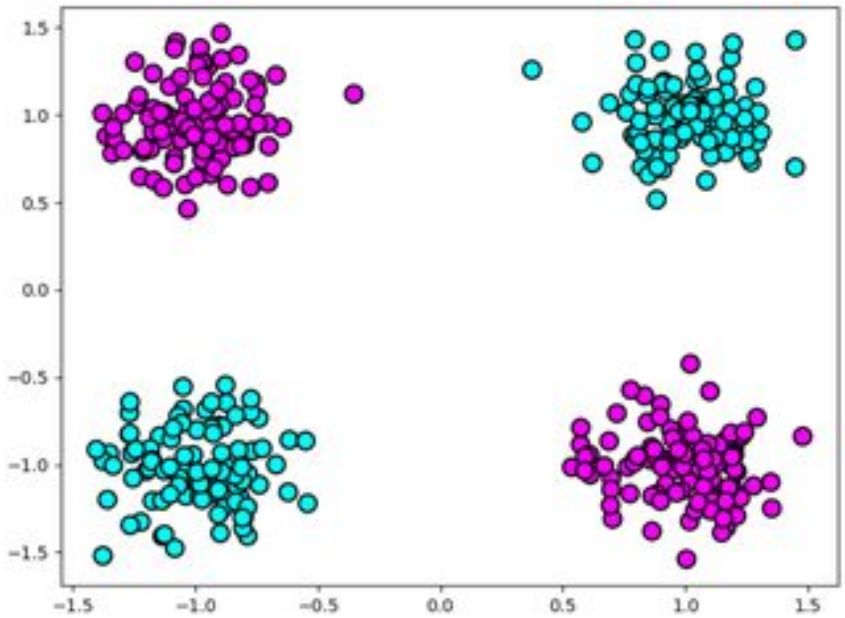



That is, rotated data may result in a completely different model.

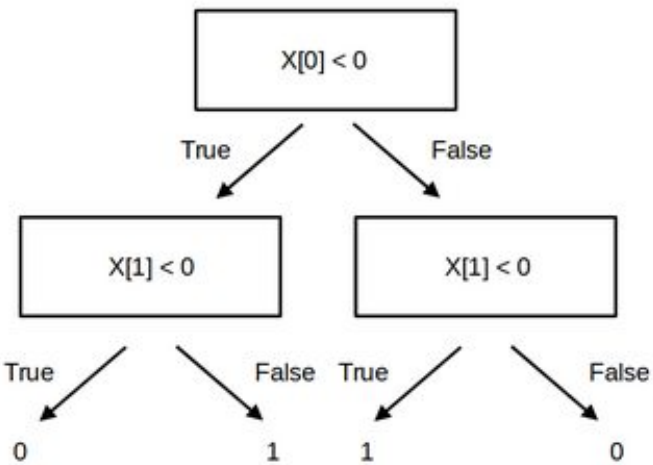
That is why decision trees are often preceded by the *principal component analysis (PCA)* transformation, which aligns data along the axes of maximum data variance.

# XOR Training Problem

Consider the following training dataset:

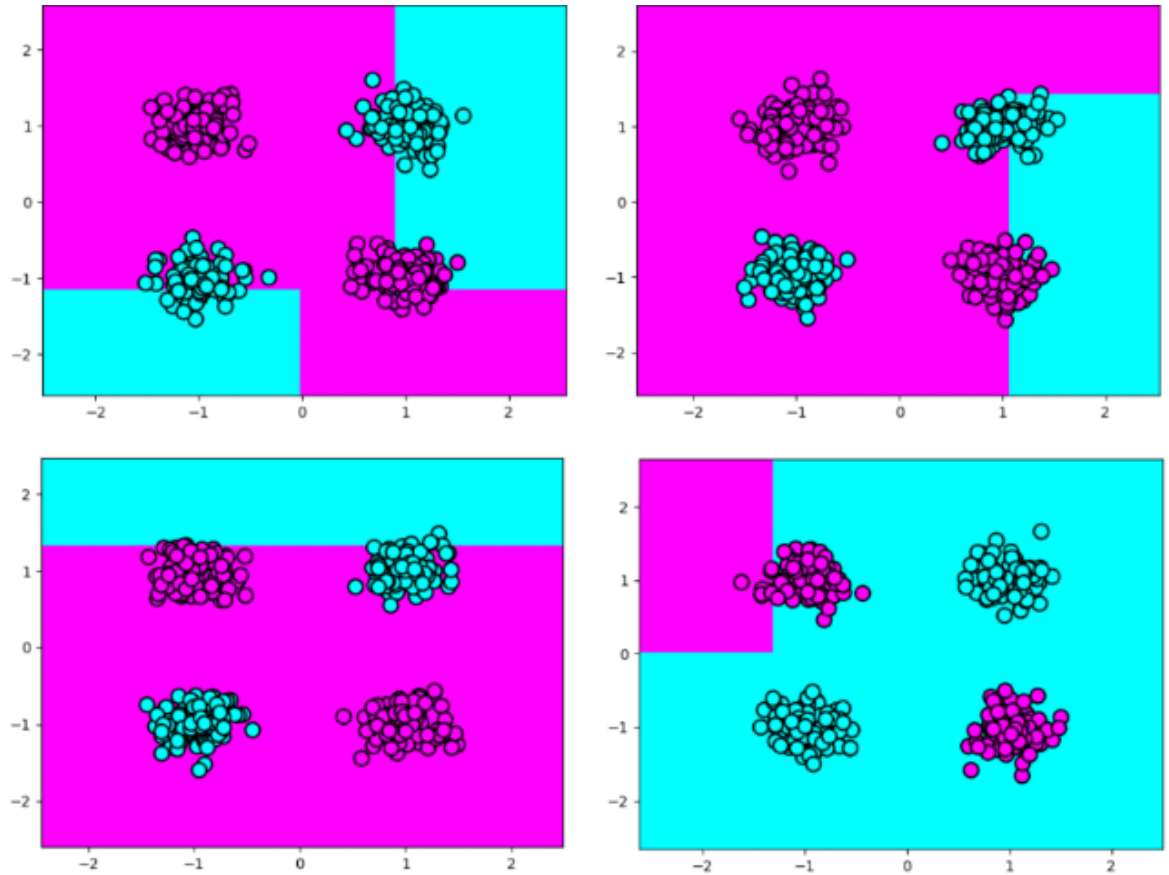



An ideal decision tree would look like this:



# Attempts at Training on XOR

Max depth = 2:

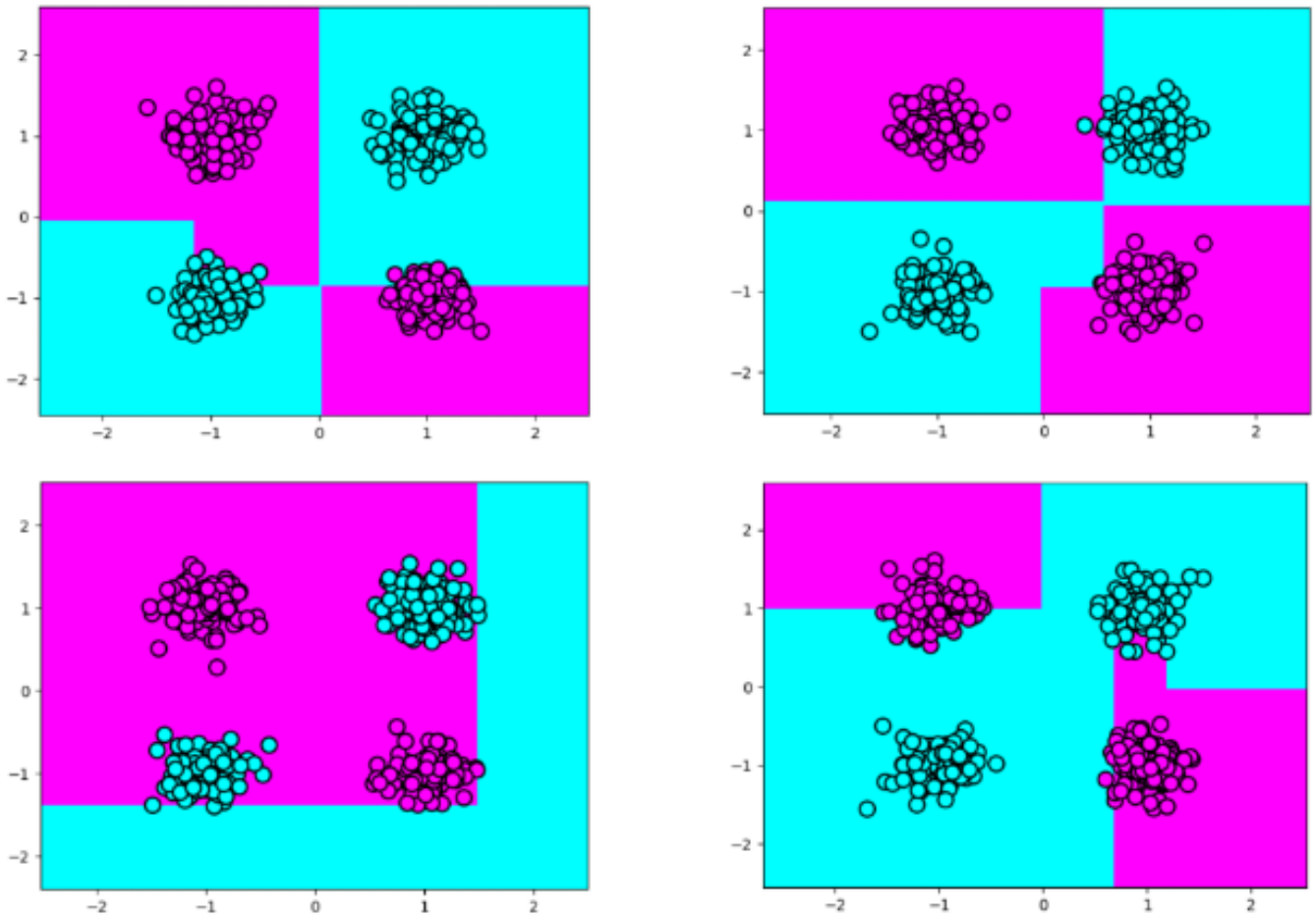


The problem: Both information gain and decrease in impurity consider only the relationship of a *single* attribute and the class.

However, there is no relationship between a single attribute and the class; both attributes need to be considered together!

# More Attempts at Training on XOR

Max depth = 3:



It's better but still fails occasionally.

# Advantages of Decision Trees

- Simple to understand and interpret; trees can be visualized.
- Uses a white box model, where conditions are easily explained by boolean logic.
- Can approximate an arbitrary (reasonable) boundary and capture complex non-linear relationships between attributes.
- Capable of handling multi-class problems.
- Little data preparation, unlike other techniques requiring normalization, dummy variables, or missing value removal.
- Handles numerical and categorical data.
- Not sensitive to outliers since the splitting is based on the proportion of examples within the split ranges and not on absolute values.
- The cost of using a well-balanced tree is logarithmic in the number of data points used to train it.

# Disadvantages of Decision Trees

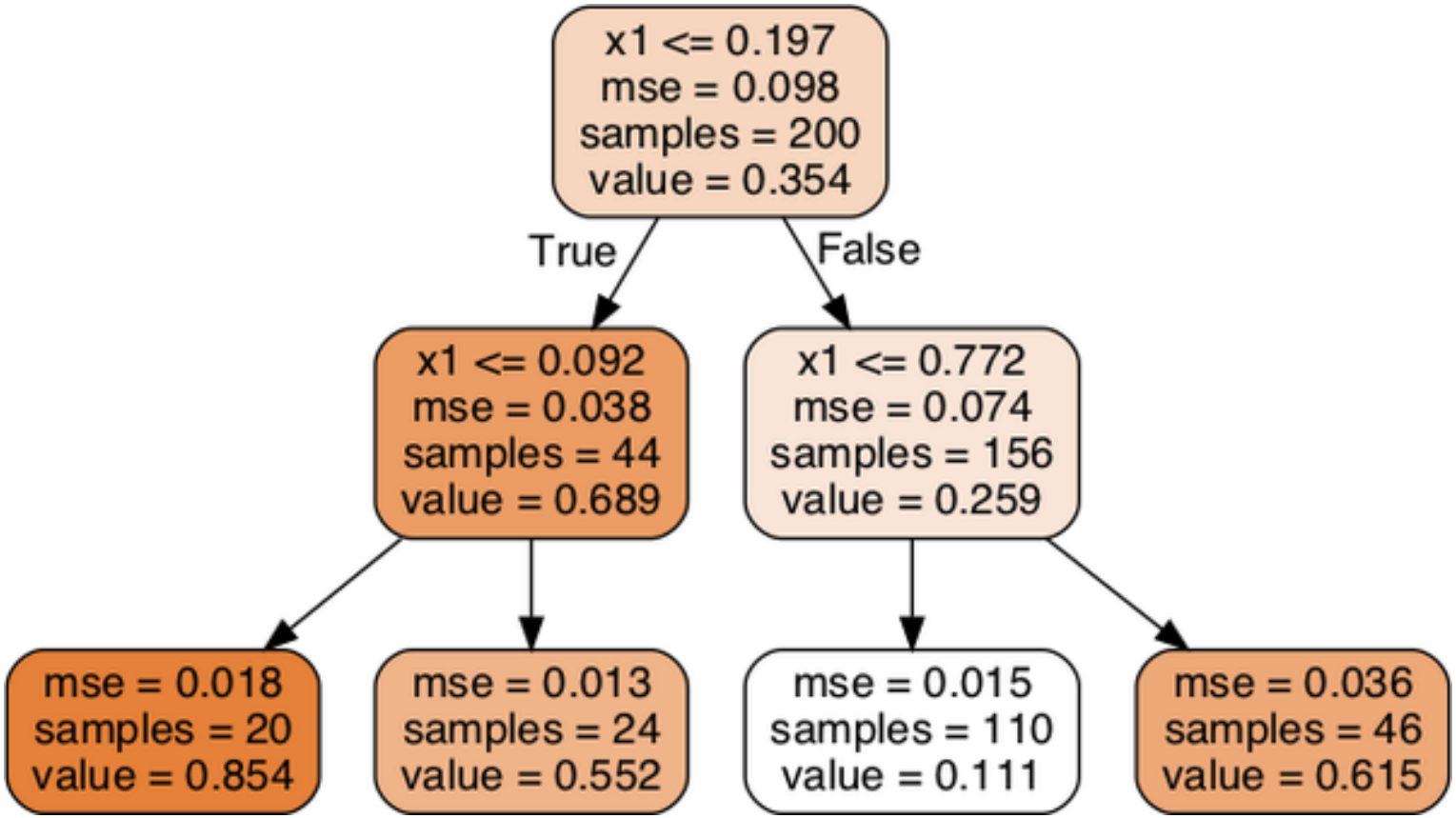
- Overfitting: Trees can be over-complex and not generalize well, needing pruning or limits on tree depth.
- Instability: Small data variations can result in very different trees. This is mitigated in ensemble methods.
- Non-smooth predictions: Decision trees make piecewise constant approximations, which are not suitable for extrapolation.
- Difficulty expressing certain concepts, such as XOR, parity, or multiplexer problems (see the next slide).
- Bias in trees: Decision trees can create biased trees if some classes dominate. Balancing the dataset is recommended.
- Learning optimal trees is NP-complete: Heuristic algorithms like greedy algorithms are used, which do not guarantee globally optimal trees. Ensemble methods can help.

# History of Decision Trees

- Hunt and colleagues use exhaustive search decision-tree methods (CLS) to model human concept learning in the 1960's.
- In the late 70's, Quinlan developed ID3 with the information gain heuristic to learn expert systems from examples.
- Simultaneously, Breiman, Friedman, and colleagues develop CART (Classification and Regression Trees), similar to ID3.
- In the 1980s, various improvements were introduced to handle noise, continuous features, missing features, and improved splitting criteria. Various expert-system development tools results.
- Quinlan's updated decision-tree package (C4.5) released in 1993.

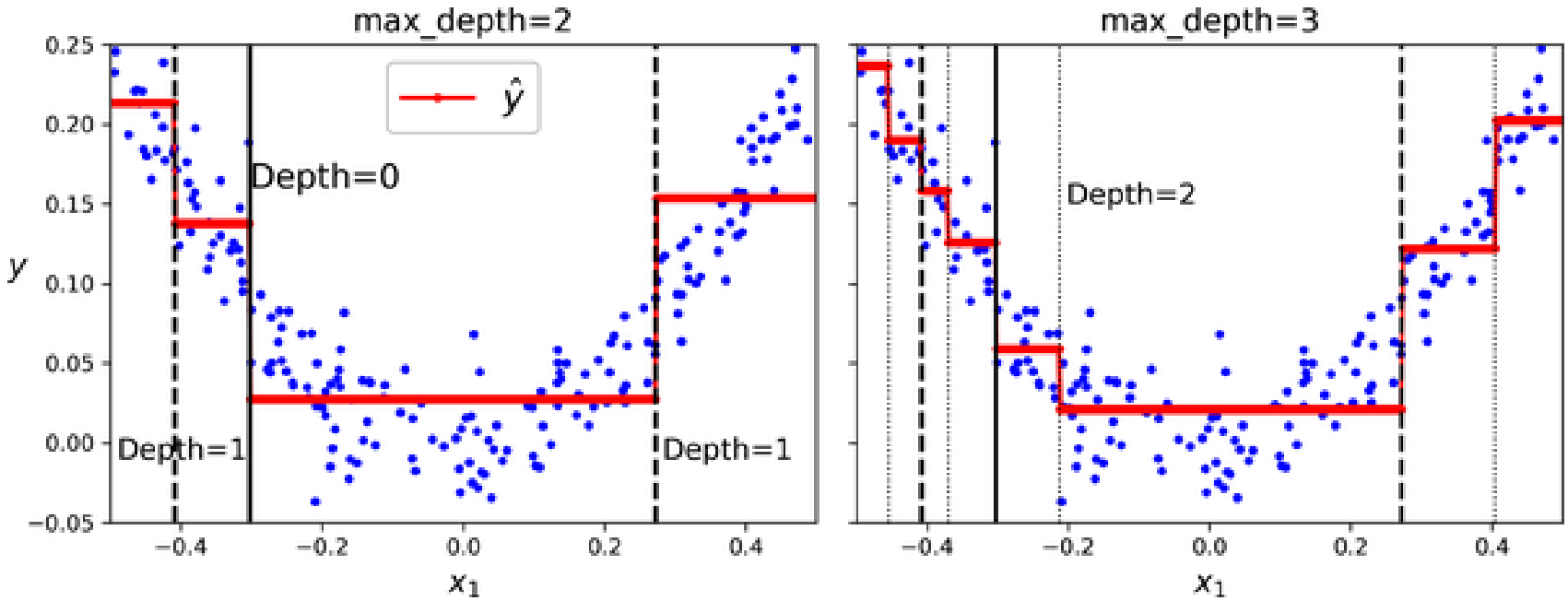
# Comment on Regression Trees

Decision trees can also be used to approximate functions. Assign a function value to the leaves instead of classes.




Here, "mse" is the mean-squared-error.

# Comment on Regression Trees




Intuitively, for every subinterval of $x_1$, the value (the red line) is at the average $y$ over the subinterval.

How are the subintervals being set?

# Regression Trees

A *regression tree* is a decision tree whose leaves are labeled by values from $\mathbb{R}$.

We follow the same procedure as in decision trees during inference on an input $\vec{x}$.

How exactly are such trees trained?

We aim to minimize the mean squared error.

Assume, for the moment, that we want to train a single-node tree. What will be the best labeling value for the single node?

Now, consider the training for arbitrary trees.

Assume ordinal attributes.

The algorithm also works for discrete attributes; ordinal attributes, however, allow us to make binary splits.

The training procedure is the same as for the decision trees, except that the splits and cut points are selected differently.

# Regression Trees

Given a dataset $\mathcal{D} = \{(\vec{x}_1, d_1), \ldots, (\vec{x}_p, d_p)\}$, we denote by $\bar{\mathcal{D}}$ the average *desired* value in $\mathcal{D}$, that is $\bar{\mathcal{D}} = \frac{1}{p}\sum_{k=1}^{p} d_k$.

Consider a dataset $\mathcal{D}$ and let us find a cut point for $A$ in $\mathcal{D}$.

We are looking for a value $H$ of the attribute $A$ such that the split:

$$\mathcal{D}_{\leq H} = \{(\vec{x}, d) \in \mathcal{D} \mid A(\vec{x}) \leq H\} \quad \mathcal{D}_{>H} = \{(\vec{x}, d) \in \mathcal{D} \mid A(\vec{x}) > H\}$$

Minimizes the following *split error*:

$$\frac{1}{|\mathcal{D}_{\leq H}|} \sum_{(\vec{x},d)\in\mathcal{D}_{\leq H}} \left(d - \bar{\mathcal{D}}_{\leq H}\right)^2 + \frac{1}{|\mathcal{D}_{>H}|} \sum_{(\vec{x},d)\in\mathcal{D}_{>H}} \left(d - \bar{\mathcal{D}}_{>H}\right)^2$$

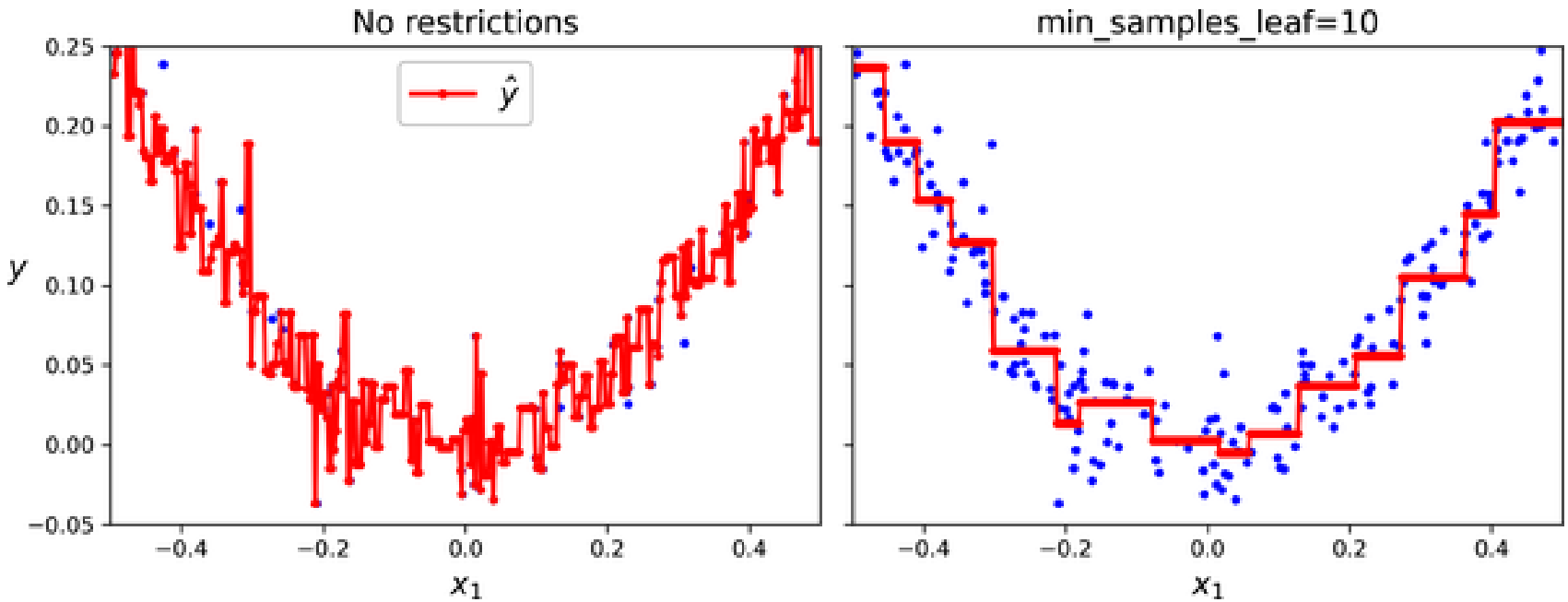
Denote by MinE($\mathcal{D}$) the minimum of the split error over all attributes $A$ and all cut points $H$.Compute

$$\Delta = \left(\frac{1}{|\mathcal{D}|} \sum_{(\vec{x},d)\in\mathcal{D}} \left(d - \bar{\mathcal{D}}\right)^2\right) - \text{MinE}$$

If $\Delta$ is large enough, split on $A$ and $H$ that minimize the split error. Otherwise, stop splitting and label the leaf with $\bar{D}$.

# Regression Tress

Without any lower bound on the number of examples in the leaves, the algorithm will eventually overfit by splitting into (possibly) singleton leaves.



# Probabilistic Classification

# Probabilistic Classification – Idea

Imagine that

- I look out of a window and see a bird,
- it is black, approx. 25 cm long and has a rather yellow beak.

My daughter asks: What kind of bird is this?

My usual answer: This is *probably* a kind of blackbird (kos černý in Czech).

Here *probably* means that out of my extensive catalog of four kinds of birds that I can recognize, "blackbird" gets the highest degree of belief based on *features* of this particular bird.

Frequentists might say that the largest proportion of birds with similar features I have ever seen were blackbirds.

The degree of belief (Bayesian), or the relative frequency (frequentists), is the *probability*.

# Basic Discrete Probability Theory

- A finite or countably infinite set $\Omega$ of *possible outcomes*, $\Omega$ is called *sample space*.

  Experiment: Roll one dice once. Sample space: $\Omega = \{1, \ldots, 6\}$
- Each element $\omega$ of $\Omega$ is assigned a "probability" value $f(\omega)$, here $f$ must satisfy
  - $f(\omega) \in [0, 1]$ for all $\omega \in \Omega$,
  - $\sum_{\omega \in \Omega} f(\omega) = 1$.

  If the dice is fair, then $f(\omega) = \frac{1}{6}$ for all $\omega \in \{1, \ldots, 6\}$.
- An *event* is any subset $E$ of $\Omega$.
- The *probability* of a given event $E \subseteq \Omega$ is defined as

$$P(E) = \sum_{\omega \in E} f(\omega)$$

  Let $E$ be the event that an odd number is rolled, i.e., $E = \{1, 3, 5\}$. Then $P(E) = \frac{1}{2}$.
- Basic laws: $P(\Omega) = 1$, $P(\emptyset) = 0$, given disjoint sets $A, B$ we have $P(A \cup B) = P(A) + P(B)$, $P(\Omega \setminus A) = 1 - P(A)$.

# Conditional Probability and Independence

- $P(A \mid B)$ is the probability of $A$ given $B$ (assume $P(B) > 0$) defined by

$$P(A \mid B) = P(A \cap B)/P(B)$$

(We assume that $B$ is all and only information known.)

A fair dice: what is the probability that 3 is rolled assuming that an odd number is rolled? ... and assuming that an even number is rolled?

- $A$ and $B$ are independent if $P(A \cap B) = P(A) \cdot P(B)$.

It is easy to show that if $P(B) > 0$, then

$A$, $B$ are independent iff $P(A \mid B) = P(A)$.

# Random Variables and Random Vectors

- A *random variable* $X$ is a function $X : \Omega \to \mathbb{R}$.
  A dice: $X : \{1, \ldots, 6\} \to \{0, 1\}$ such that $X(n) = n \mod 2$.
- A *random vector* is a function $X : \Omega \to \mathbb{R}^d$.
  We use $X = (X_1, \ldots, X_d)$ where $X_i$ is a random variable returning the $i$-th component of $X$.
- Consider random variables $X_1, X_2$ and $Y$. The variables $X_1, X_2$ are *conditionally independent given $Y$* if for all $x_1, x_2$ and $y$ we have that

$$P(X_1 = x_1, X_2 = x_2 \mid Y = y) = P(X_1 = x_1 \mid Y = y) \cdot P(X_2 = x_2 \mid Y = y)$$

# Random Vectors – Example

Let $\Omega$ be a space of colored geometric shapes that are divided into two categories ($\mathbf{1}$ and $\mathbf{0}$).

Assume a random vector $X = (X_{color}, X_{shape}, X_{cat})$ where

- $X_{color} : \Omega \to \{red, blue\}$,
- $X_{shape} : \Omega \to \{circle, square\}$,
- $X_{cat} : \Omega \to \{\mathbf{1}, \mathbf{0}\}$.

The following tables give probability distribution of values:

category $\mathbf{1}$:

| | circle | square |
|---|---|---|
| red | 0.2 | 0.02 |
| blue | 0.02 | 0.01 |

category $\mathbf{0}$:

| | circle | square |
|---|---|---|
| red | 0.05 | 0.3 |
| blue | 0.2 | 0.2 |

# Random Vectors – Example

Example:
$P(\mathit{red}, \mathit{circle}, \mathbf{1}) = P(X_{\mathit{color}} = \mathit{red}, X_{\mathit{shape}} = \mathit{circle}, X_{\mathit{cat}} = \mathbf{1}) = 0.2$

"Summing over" all possible values of some variable(s) gives the distribution of the rest:

$$\begin{aligned} P(\mathit{red}, \mathit{circle}) &= P(X_{\mathit{color}} = \mathit{red}, X_{\mathit{shape}} = \mathit{circle}) \\ &= P(\mathit{red}, \mathit{circle}, \mathbf{1}) + P(\mathit{red}, \mathit{circle}, \mathbf{0}) \\ &= 0.2 + 0.05 = 0.25 \end{aligned}$$

$$P(\mathit{red}) = 0.2 + 0.02 + 0.05 + 0.3 = 0.57$$

Thus also, all conditional probabilities can be computed:

$$P(\mathbf{1} \mid \mathit{red}, \mathit{circle}) = \frac{P(\mathit{red}, \mathit{circle}, \mathbf{1})}{P(\mathit{red}, \mathit{circle})} = \frac{0.2}{0.25} = 0.8$$

# Bayesian Classification

Let $\Omega$ be a sample space (a universum) of all objects that can be classified. We assume a probability $P$ on $\Omega$.

We consider the problem of **binary classification**:

- Let $Y$ be the random variable for the category which takes values in $\{\mathbf{0}, \mathbf{1}\}$.
- Let $X$ be the random vector describing $n$ features of a given instance, i.e., $X = (X_1, \ldots, X_n)$
  - Denote by $\vec{x} \in \mathbb{R}^n$ values of $X$,
  - and by $x_i \in \mathbb{R}$ values of $X_i$.

**Bayes classifier:** Given a vector of feature values $\vec{x}$,

$$C^{Bayes}(\vec{x}) := \begin{cases} \mathbf{1} & \text{if } P(Y = \mathbf{1} \mid X = \vec{x}) \geq P(Y = \mathbf{0} \mid X = \vec{x}) \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

Intuitively, $C^{Bayes}$ assigns to $\vec{x}$ the most probable category it might be in.

# Bayesian Classification – Example

Imagine a conveyor belt with apples and apricots.

A machine is supposed to correctly distinguish apples from apricots based on their weight and diameter.

That is,

- $Y \in \{\mathbf{1}, \mathbf{0}\}$
  (here our interpretation is $\mathbf{1}$ = apple, $\mathbf{0}$ = appricot)
- $X = (X_{weight}, X_{diam})$

We are given a fruit of a diameter of $5cm$ that weighs $40g$.

The Bayes classifier compares $P(Y = \mathbf{1} \mid X = (40g, 5cm))$ with $P(Y = \mathbf{0} \mid X = (40g, 5cm))$ and selects the more probable category given the features.

**Crucial question:** Is such a classifier good?

There are other classifiers, e.g., one which compares the weight divided by 10 with the diameter and decides based on the answer, or maybe a classifier that sums the weight and the diameter and compares the result with a constant, etc.

# Bayes Classifier

Let $C$ be an arbitrary *classifier*, that is a function that to every feature vector $\vec{x} \in \mathbb{R}^n$ assigns a class from $\{\mathbf{0}, \mathbf{1}\}$.

Define the error of the classifier $C$ by

$$E_C = P(Y \neq C)$$

(Here we slightly abuse notation and apply $C$ to samples, technically we apply the composition $C \circ X$ of $C$ and $X$ which first determines the features using $X$ and then classifies according to $C$).

## Theorem

*The Bayes classifier $C^{Bayes}$ minimizes $E_C$, that is*

$$E_{C^{Bayes}} = \min_{C \text{ is a classifier}} E_C$$

# Practical Use of Bayes Classifier

**The crucial problem:** The probability $P$ is not known!
In particular, where to get $P(Y = \mathbf{1} \mid X = \vec{x})$ ?
Note that $P(Y = \mathbf{0} \mid X = \vec{x}) = 1 - P(Y = \mathbf{1} \mid X = \vec{x})$

Given no other assumptions, this requires a table showing the probability of the category $\mathbf{1}$ for each possible feature vector $\vec{x}$.

Where do you get these probabilities?

In some cases, the probabilities might come from the knowledge of the solved problem (e.g., applications in physics might be supported by a theory giving the probabilities).

In most cases, however, $P$ is estimated from sampled data by

$$\bar{P}(Y = \mathbf{1} \mid X = \vec{x}) = \frac{\text{number of samples with } Y = \mathbf{1} \text{ and } X = \vec{x}}{\text{number of samples with } X = \vec{x}}$$

(We use $\bar{P}$ to denote an estimate of $P$ from data.)

# Estimating $P$

Consider a problem with $X = (X_1, X_2, X_3)$ where each $X_i$ returns either 0 or 1. What might the data look like?

Part of the data table:

| $Y$ | $X_1$ | $X_2$ | $X_3$ |
|---|---|---|---|
| **1** | 1 | 0 | 1 |
| **1** | 0 | 1 | 1 |
| **0** | 1 | 0 | 1 |
| **0** | 0 | 0 | 1 |
| **1** | 0 | 0 | 0 |
| **0** | 1 | 1 | 1 |
| ... | | | |

All data with $X_1 = 1$, $X_2 = 0$, $X_3 = 1$:

| $Y$ | $X_1$ | $X_2$ | $X_3$ |
|---|---|---|---|
| **1** | 1 | 0 | 1 |
| **1** | 1 | 0 | 1 |
| **0** | 1 | 0 | 1 |
| **0** | 1 | 0 | 1 |
| **1** | 1 | 0 | 1 |
| **1** | 1 | 0 | 1 |

Estimate: $\bar{P}(\mathbf{1} \mid 1, 0, 1) = 2/3$

The probability table and the necessary data are typically too large!

Concretely, if all $X_1, \ldots, X_n$ are binary, there are $2^n$ probabilities $P(Y = \mathbf{1} \mid X = \vec{x})$, one for each possible $\vec{x} \in \{0, 1\}^n$.

# Let's Look at It the Other Way Round

**Theorem (Bayes,1764)**

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

**Proof.**

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{\frac{P(A \cap B)}{P(A)} \cdot P(A)}{P(B)} = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

□

# Bayesian Classification

Determine the category for $\vec{x}$ by computing

$$P(Y = y \mid X = \vec{x}) = \frac{P(Y = y) \cdot P(X = \vec{x} \mid Y = y)}{P(X = \vec{x})}$$

for both $y \in \{\mathbf{0}, \mathbf{1}\}$ and deciding whether or not the following holds:

$$P(Y = \mathbf{1} \mid X = \vec{x}) \geq P(Y = \mathbf{0} \mid X = \vec{x})$$

So, to make the classifier, we need to compute the following:

- The prior $P(Y = \mathbf{1})$ (then $P(Y = \mathbf{0}) = 1 - P(Y = \mathbf{1})$)
- The conditionals $P(X = \vec{x} \mid Y = y)$ for $y \in \{\mathbf{0}, \mathbf{1}\}$ and for every $\vec{x}$

# Estimating the Prior and Conditionals

- $P(Y = \mathbf{1})$ can be easily estimated from data by

$$\bar{P}(Y = \mathbf{1}) = \frac{\text{number of samples with } Y = \mathbf{1}}{\text{number of all samples}}$$

- If the dimension of features is small, $P(X = \vec{x} \mid Y = y)$ can be estimated from data similarly as $P(Y = \mathbf{1} \mid X = \vec{x})$ by

$$\bar{P}(X = \vec{x} \mid Y = y) = \frac{\text{number of samples with } Y = y \text{ and } X = \vec{x}}{\text{number of samples with } Y = y}$$

Unfortunately, for higher dimensional data too many samples are needed to estimate all $P(X = \vec{x} \mid Y = y)$ (there are too many $\vec{x}$'s).

So where is the advantage of using the Bayes thm.??

We introduce *independence assumptions* about the features!

# Naive Bayes

- We assume that features are (conditionally) independent *given the category*. That is for all $\vec{x} = (x_1, \ldots, x_n)$ and $y \in \{\mathbf{0}, \mathbf{1}\}$ we **assume**:

$$P(X = x \mid Y = y) = P(X_1 = x_1, \cdots, X_n = x_n \mid Y)$$
$$= \prod_{i=1}^{n} P(X_i = x_i \mid Y = y)$$

- Therefore, we only need to specify $P(X_i = x_i \mid Y = y)$ for each possible pair of a feature-value $x_i$ and $y \in \{\mathbf{0}, \mathbf{1}\}$.

  Note that if all $X_i$ are binary (values in $\{0, 1\}$), this requires specifying only $2n$ parameters:

  $$P(X_i = 1 \mid Y = \mathbf{1}) \text{ and } P(X_i = 1 \mid Y = \mathbf{0}) \text{ for each } X_i$$

  as $P(X_i = 0 \mid Y = y) = 1 - P(X_i = 1 \mid Y = y)$ for $y \in \{\mathbf{0}, \mathbf{1}\}$.

Compared to specifying $2^n$ parameters without any independence assumption.

# Estimating the marginal probabilities

Estimate the probabilities $P(X_i = x_i \mid Y = y)$ by

$$\bar{P}(X_i = x_i \mid Y = y) = \frac{\text{number of samples with } X_i = x_i \text{ and } Y = y}{\text{number of samples with } Y = y}$$

**Example:** Consider a problem with $X = (X_1, X_2, X_3)$ where each $X_i$ returns either 0 or 1. The data is

| $Y$ | $X_1$ | $X_2$ | $X_3$ |
|---|---|---|---|
| **1** | 1 | 0 | 1 |
| **1** | 0 | 1 | 1 |
| **0** | 1 | 0 | 1 |
| **0** | 0 | 0 | 1 |
| **1** | 0 | 0 | 0 |
| **0** | 1 | 1 | 1 |

$\bar{P}(X_1 = 1 \mid Y = \mathbf{1}) = 1/3$ $\quad$ $\bar{P}(X_1 = 1 \mid Y = \mathbf{0}) = 2/3$

$\bar{P}(X_2 = 1 \mid Y = \mathbf{1}) = 1/3$ $\quad$ $\bar{P}(X_2 = 1 \mid Y = \mathbf{0}) = 1/3$

$\bar{P}(X_3 = 1 \mid Y = \mathbf{1}) = 2/3$ $\quad$ $\bar{P}(X_3 = 1 \mid Y = \mathbf{0}) = 1$

# Naive Bayes – Example

Consider classification of geometric shapes:

$X_1 \in \{small, medium, large\}$

$X_2 \in \{red, blue, green\}$

$X_3 \in \{square, triangle, circle\}$

Assume that we have already estimated the following probabilities:

| | $Y = \mathbf{1}$ | $Y = \mathbf{0}$ |
|---|---|---|
| $\bar{P}(Y)$ | 0.5 | 0.5 |
| $\bar{P}(small \mid Y)$ | 0.4 | 0.4 |
| $\bar{P}(medium \mid Y)$ | 0.1 | 0.2 |
| $\bar{P}(large \mid Y)$ | 0.5 | 0.4 |
| $\bar{P}(red \mid Y)$ | 0.9 | 0.3 |
| $\bar{P}(blue \mid Y)$ | 0.05 | 0.3 |
| $\bar{P}(green \mid Y)$ | 0.05 | 0.4 |
| $\bar{P}(square \mid Y)$ | 0.05 | 0.4 |
| $\bar{P}(triangle \mid Y)$ | 0.05 | 0.3 |
| $\bar{P}(circle \mid Y)$ | 0.9 | 0.3 |

Does $(medium, red, circle)$ belong to the category $\mathbf{1}$ ?

| | $Y = \mathbf{1}$ | $Y = \mathbf{0}$ |
|---|---|---|
| $\bar{P}(Y)$ | 0.5 | 0.5 |
| $\bar{P}(medium \mid Y)$ | 0.1 | 0.2 |
| $\bar{P}(red \mid Y)$ | 0.9 | 0.3 |
| $\bar{P}(circle \mid Y)$ | 0.9 | 0.3 |

Denote $\vec{x} = (medium, red, circle)$.

$$P(Y = \mathbf{1} \mid X = \vec{x}) =$$
$$= P(\mathbf{1}) \cdot P(medium \mid \mathbf{1}) \cdot P(red \mid \mathbf{1}) \cdot P(circle \mid \mathbf{1}) \,/\, P(X = \vec{x})$$
$$\doteq 0.5 \cdot 0.1 \cdot 0.9 \cdot 0.9 \,/\, P(X = \vec{x}) = 0.0405 / P(X = \vec{x})$$

$$P(Y = \mathbf{0} \mid X = \vec{x}) =$$
$$= P(\mathbf{0}) \cdot P(medium \mid \mathbf{0}) \cdot P(red \mid \mathbf{0}) \cdot P(circle \mid \mathbf{0}) \,/\, P(X = \vec{x})$$
$$\doteq 0.5 \cdot 0.2 \cdot 0.3 \cdot 0.3 \,/\, P(X = \vec{x}) = 0.009 / P(X = \vec{x})$$

(Note that we used the estimates $\bar{P}$ of $P$ to finish the computation above.)
Apparently,

$$P(Y = \mathbf{1} \mid X = \vec{x}) \doteq 0.0405 / P(X = \vec{x}) > 0.009 / P(X = \vec{x}) \doteq P(\mathbf{0} \mid X = \vec{x})$$

So we classify $\vec{x}$ to the category $\mathbf{1}$.

# Estimating Probabilities in Practice

We already know that $P(X_i = x_i \mid Y = y)$ can be estimated by

$$\bar{P}(X_i = x_i \mid Y = y) = \ell_{y,x_i} \,/\, \ell_y$$

where

- $\ell_{y,x_i}$ = number of samples with $Y = y$ and $X_i = x_i$
- $\ell_y$ = number of samples with $Y = y$

**Problem:** If, by chance, a rare value $x_i$ of a feature $X_i$ never occurs in the training data, we get

$$\bar{P}(X_i = x_i \mid Y = y) = 0 \quad \text{for both } y \in \{\mathbf{0}, \mathbf{1}\}$$

But then $\bar{P}(X = x) = 0$ for $x$ containing the value $x_i$ for $X_i$, and thus $\bar{P}(Y = y \mid X = x)$ is not well defined.
Moreover, $\bar{P}(Y = y) \cdot \bar{P}(X = x \mid Y = y) = 0$ (for $y \in \{\mathbf{0}, \mathbf{1}\}$) so even this cannot be used for classification.

# Probability Estimation Example

Training data:

| Size | Color | Shape | Class |
|---|---|---|---|
| small | red | circle | **1** |
| large | red | circle | **1** |
| small | red | triangle | **0** |
| large | blue | circle | **0** |

Estimated probabilities:

| | $Y = \mathbf{1}$ | $Y = \mathbf{0}$ |
|---|---|---|
| $\bar{P}(Y)$ | 0.5 | 0.5 |
| $\bar{P}(small \mid Y)$ | 0.5 | 0.5 |
| $\bar{P}(medium \mid Y)$ | 0 | 0 |
| $\bar{P}(large \mid Y)$ | 0.5 | 0.5 |
| $\bar{P}(red \mid Y)$ | 1 | 0.5 |
| $\bar{P}(blue \mid Y)$ | 0 | 0.5 |
| $\bar{P}(green \mid Y)$ | 0 | 0 |
| $\bar{P}(square \mid Y)$ | 0 | 0 |
| $\bar{P}(triangle \mid Y)$ | 0 | 0.5 |
| $\bar{P}(circle \mid Y)$ | 1 | 0.5 |

Note that $\bar{P}(medium \mid \mathbf{1}) = P(medium \mid \mathbf{0}) = 0$ and thus also $\bar{P}(medium, red, circle) = 0$.

So what is $\bar{P}(\mathbf{1} \mid medium, red, circle)$ ?

# Smoothing

- To account for estimation from small samples, probability estimates are adjusted or *smoothed*.
- *Laplace smoothing* adds one to every count of feature values

$$\tilde{P}(X_i = x_i \mid Y = y) = \frac{\ell_{y,x_i} + 1}{\ell_y + v_i}$$

where

- $\ell_y$ = number of training samples with $Y = y$,
- $\ell_{y,x_i}$ = number of training samples with $Y = y$ and $X_i = x_i$,
- $v_i$ is the number of all distinct values of the variable $X_i$.

To understand note that

$$\ell_y = \sum_{x_i \text{ is a value of } X_i} \ell_{y,x_i}$$

and thus

$$\bar{P}(X_i = x_i \mid Y = y) = \ell_{y,x_i} \,/ \sum_{x_i \text{ is a value of } X_i} \ell_{y,x_i}$$

$$\tilde{P}(X_i = x_i \mid Y = y) = (\ell_{y,x_i} + 1) \,/ \sum_{x_i \text{ is a value of } X_i} (\ell_{y,x_i} + 1)$$

# Laplace Smoothing Example

- Assume training set contains 10 samples of category **1**:
  - 4 small
  - 0 medium
  - 6 large
- Estimate parameters as follows
  - $\tilde{P}(\textit{small} \mid \mathbf{1}) = (4+1)/(10+3) = 0.384$
  - $\tilde{P}(\textit{medium} \mid \mathbf{1}) = (0+1)/(10+3) = 0.0769$
  - $\tilde{P}(\textit{large} \mid \mathbf{1}) = (6+1)/(10+3) = 0.538$

# Continuous Features

$\Omega$ may be (potentially) continuous, $X_i$ may assign a continuum of values in $\mathbb{R}$.

- The probabilities are computed using *probability density* $p : \mathbb{R} \to \mathbb{R}^+$.
  A random variable $X : \Omega \to \mathbb{R}^+$ has a density $p : \mathbb{R} \to \mathbb{R}^+$ if for every interval $[a, b]$ we have

$$P(a \leq X \leq b) = \int_a^b p(x)dx$$

  Usually, $P(X_i \mid Y = y)$ is used to denote the *density* of $X_i$ conditioned on $Y = y$.
- The densities $P(X_i \mid Y = y)$ are usually estimated using Gaussian densities as follows:
  - Estimate the mean $\mu_{iy}$ and the standard deviation $\sigma_{iy}$ based on training data.
  - Then put

$$\bar{P}(X_i \mid Y = y) = \frac{1}{\sigma_{iy}\sqrt{2\pi}} \exp\left(\frac{-(X_i - \mu_{iy})^2}{2\sigma_{iy}^2}\right)$$

# Comments on Naive Bayes

- Tends to work well despite rather a strong assumption of conditional independence of features.
- Experiments show that it is quite competitive with other classification methods.

  Even if the probabilities are not accurately estimated, it often picks the correct maximum probability category.
- Directly constructs a model from parameter estimates that are calculated from the training data.
- Typically handles outliers and noise well.
- Missing values are easy to deal with (simply average overall missing values in feature vectors).

# Bayesian Networks (Basic Information)

In the Naive Bayes, we have assumed that *all* features $X_1, \ldots, X_n$ are independent.

This is usually not realistic.

E.g. Variables "rain" and "grass wet" are (usually) strongly dependent.

What if we return some dependencies?

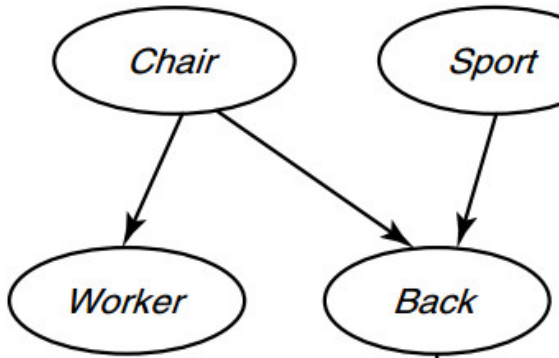
(But now in a well-defined sense.)

Bayesian networks are a graphical model that uses a directed acyclic graph to specify dependencies among variables.

# Bayesian Networks – Example

| P(C = T) | P(C = F) |
|---|---|
| 0.8 | 0.2 |

| P(S = T) | P(S = F) |
|---|---|
| 0.02 | 0.98 |




| C | P(W = T\|C) | P(W = F\|C) |
|---|---|---|
| T | 0.9 | 0.1 |
| F | 0.01 | 0.99 |

| C | S | P(B = T\|C,S) | P(B = F\|C,S) |
|---|---|---|---|
| T | T | 0.9 | 0.1 |
| T | F | 0.2 | 0.8 |
| F | T | 0.9 | 0.1 |
| F | F | 0.01 | 0.99 |

| B | P(A = T\|B) | P(A = F\|B) |
|---|---|---|
| T | 0.7 | 0.3 |
| F | 0.1 | 0.9 |

Now, e.g.,

$$P(C, S, W, B, A) = P(C) \cdot P(S) \cdot P(W \mid C) \cdot P(B \mid C, S) \cdot P(A \mid B)$$

Now, we may, e.g., infer the probability $P(C = T \mid A = T)$ that we sit in the wrong chair, assuming that our back aches.

We have to store only 10 numbers as opposed to $2^5 - 1$ possible probabilities for all vectors of values of $C, S, W, B, A$.

# Bayesian Networks – Learning & Naive Bayes

Many algorithms have been developed for learning:

- the structure of the graph of the network,
- the *conditional probability tables*.

The methods are based on maximum-likelihood estimation, gradient descent, etc.

Automatic procedures are usually combined with expert knowledge.

---

Can you express the naive Bayes for $Y, X_1, \ldots, X_n$ using a Bayesian network?

# Classifier Evaluation

# Classifier

Assume binary classification into two classes $\{0, 1\}$.

Consider a classification dataset:

$$\{(\vec{x}_k, c_k) \mid k = 1, \ldots, p\}$$

Here $\vec{x}_k$ is a vector of attributes/features and $c_k \in \{0, 1\}$ for all $k$.

Consider a sequence of predictions generated by a classifier:

$$h_1, \ldots, h_p \in \{0, 1\}$$

Here each $h_k$ has been predicted for the $k$-the example $(\vec{x}_k, c_k)$.

How good are the predictions $h_1, \ldots, h_p$ w.r.t. $c_1, \ldots, c_p$?

There are many possible metrics ...

I will call the class 1 *positive* and the class 0 *negative*.

Note that the class 0 is not negative in the numerical sense but in the absence of something (e.g., predicted illness).

# Confusion Matrix for Binary Classifier

| | | Predicted | |
|---|---|---|---|
| | | 1 | 0 |
| Actual | 1 | TP | FN |
| | 0 | FP | TN |

- TP = number of correctly classified examples with actual class 1

$$\text{TP} = |\{k \mid h_k = 1 \wedge c_k = 1\}|$$

- TN = number of correctly classified examples with actual class 0

$$\text{TN} = |\{k \mid h_k = 0 \wedge c_k = 0\}|$$

- FP = number of incorrectly classified examples with actual class 0

$$\text{FP} = |\{k \mid h_k = 1 \wedge c_k = 0\}|$$

- FN = number of correctly classified examples with actual class 1

$$\text{FN} = |\{k \mid h_k = 0 \wedge c_k = 1\}|$$

# Example

Given a sample of 12 individuals, eight have cancer, and four are cancer-free.

Assume that we have trained a classifier with the following results:

| **Index** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Actual** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **Predicted** | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| **Result** | FN | FN | TP | TP | TP | TP | TP | TP | FP | TN | TN | TN |

| **Actual condition** | **Predicted condition** | |
|---|---|---|
| | **Cancer** | **Non-cancer** |
| **Cancer** | TP = 6 | FN = 2 |
| **Non-cancer** | FP = 1 | TN = 3 |
| **Total** | 8 + 4 = 12 | |

# Terminology

- TP aka hit
- TN aka correct rejection
- FP aka type I error, false alarm, overestimation
- FN aka type II error, miss, underestimation

Usually, TP, TN, FP, and FN are used to denote the individual examples of a particular kind and the number of these examples.

In what follows, we also use

- $P = TP + FN$ of all cases with the *actual* class 1
- $N = TN + FP$ of all cases with the *actual* class 0
- $PP = TP + FP$ of all cases with the *predicted* class 1
- $PN = TN + FN$ of all cases with the *predicted* class 0

Note that $P + N = PP + PN$ is the number of all cases.

There is a large number of derived metrics. We consider some of the most used in practice.

# Accuracy

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}}$$

Intuitively, Accuracy is the proportion of correctly classified cases w.r.t. all cases.

**Example:** Consider our cancer predictor with the confusion matrix

| **Actual condition** | **Predicted condition** | |
|---|---|---|
| | **Cancer** | **Non-cancer** |
| **Cancer** | $\text{TP} = 6$ | $\text{FN} = 2$ |
| **Non-cancer** | $\text{FP} = 1$ | $\text{TN} = 3$ |
| **Total** | $8 + 4 = 12$ | |

The Accuracy is

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}} = \frac{6 + 3}{12} = \frac{3}{4}$$

# Accuracy - Imbalanced Classes

Accuracy can be misleading when the classes are imbalanced:

- Consider 100 cases, 90 in the class 0 and 10 in the class 1,
- consider a classifier that returns 1 for a single sample of class 1 and 0 for all other samples.

| **Actual** | **Predicted** | |
|---|---|---|
| | **Pos** | **Neg** |
| **Pos** | 1 | 9 |
| **Neg** | 0 | 90 |
| **Total** | $90 + 10 = 100$ | |

The Accuracy is $91/100 > 0.9$. Pretty good, right?

However, the classifier is pretty bad in the positive cases.

In the case of cancer prediction, such a classifier would be a disaster.

# Precision & Recall

To mitigate the defect of the Accuracy, we may compute the following metrics:

$$\text{Precision} = \frac{\text{TP}}{\text{PP}} \quad (= \text{how often is predicted positive actually positive})$$

Precision is also known as positive predictive value (PPV)

$$\text{Recall} = \frac{\text{TP}}{\text{P}} \quad (= \text{how often is actually positive predicted positive})$$

Recall is also known as true positive rate, sensitivity, hit rate, and power.

# Precision & Recall - Example

**Example:** In our cancer example:

| **Actual condition** | **Predicted condition** | |
|---|---|---|
| | **Cancer** | **Non-cancer** |
| **Cancer** | TP $= 6$ | FN $= 2$ |
| **Non-cancer** | FP $= 1$ | TN $= 3$ |
| **Total** | $8 + 4 = 12$ | |

- Precision measures how often is the patient predicted to be ill truly ill (in our case, 6/7)
- Recall measures how often is an ill patient found to be ill (in our case, 6/8)

# Precision & Recall - Imbalanced Classes

- Consider 100 cases, 90 in the class 0 and 10 in the class 1,
- consider a classifier that returns 1 for a single sample of class 1 and 0 for all other samples.

| **Actual** | **Predicted** | |
|---|---|---|
| | **Pos** | **Neg** |
| **Pos** | 1 | 9 |
| **Neg** | 0 | 90 |
| **Total** | $90 + 10 = 100$ | |

$$\text{Precision} = 1$$

$$\text{Recall} = \frac{1}{10}$$

You can see that the predictor is very precise (on the class 1) but useless due to the weak Recall.

# Precision & Recall - Relative Importance

Let us get back to our cancer example:

| **Actual condition** | **Predicted condition** | |
|---|---|---|
| | **Cancer** | **Non-cancer** |
| **Cancer** | TP $= 6$ | FN $= 2$ |
| **Non-cancer** | FP $= 1$ | TN $= 3$ |
| **Total** | $8 + 4 = 12$ | |

Consider *Precision* and *Recall*.

By now, you should remember what they measure.

Which of the two is more important in medicine?

Which of the two is more important for plagiarism detectors?

Can we get a single number summarizing both Precision and Recall?

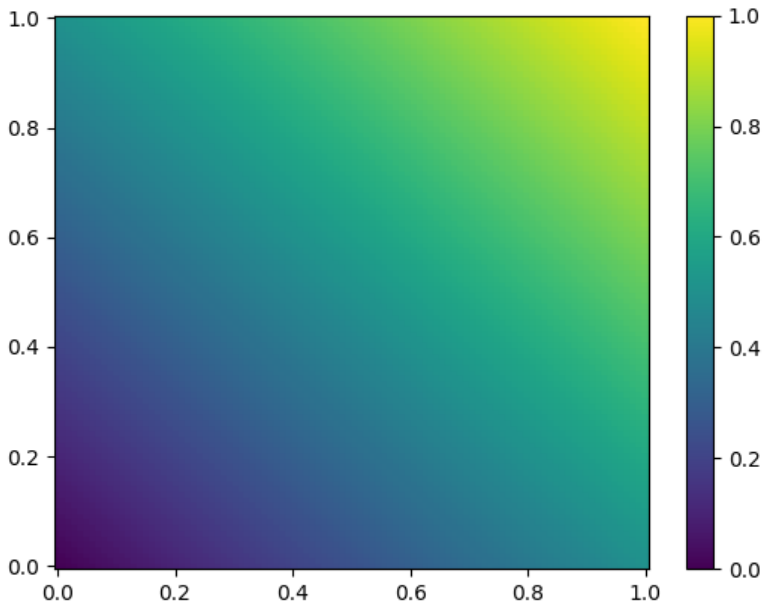
For example, to compare two classifiers.

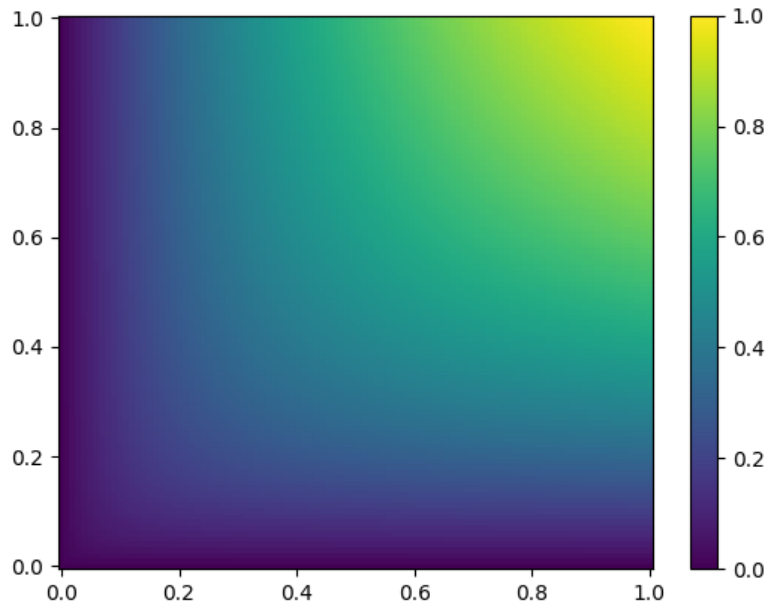
# $F_1$ Score

$F_1$ score is the harmonic mean of Recall and Precision:

$$\mathrm{F}_1 = \frac{2}{\mathrm{Recall}^{-1} + \mathrm{Precision}^{-1}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Compare the arithmetic (left) and harmonic (right) mean:




The harmonic mean prefers the two values closer to each other.

For example, the harmonic mean of 2/3 and 1/3 is (approx) 0.44444.

# $F_1$ Score - Examples

Consider the cancer example:

| **Actual condition** | **Predicted condition** | |
|---|---|---|
| | **Cancer** | **Non-cancer** |
| **Cancer** | TP = 6 | FN = 2 |
| **Non-cancer** | FP = 1 | TN = 3 |
| **Total** | 8 + 4 = 12 | |

Here $F_1 = \frac{2\text{TP}}{2\text{TP}+\text{FP}+\text{FN}} = (2 \cdot 6)/((2 \cdot 6) + 1 + 2) = 0.8$.

Our imbalanced example:

| **Actual** | **Predicted** | |
|---|---|---|
| | **Pos** | **Neg** |
| **Pos** | 1 | 9 |
| **Neg** | 0 | 90 |
| **Total** | 90 + 10 = 100 | |

Here $F_1 = \frac{2\text{TP}}{2\text{TP}+\text{FP}+\text{FN}} = (2 \cdot 1)/((2 \cdot 1) + 0 + 9) = 0.18$.

Note that the average of Precision and Recall is 0.55, which would give us a much less severe warning that the classifier is bad.

# Imbalanced Classes Once More

Note that the standard definitions of Precision and Recall for binary classifiers reveal only part of the truth.

In particular, *true negatives are not used* in the definition of $F_1$.

Consider

| **Actual** | **Predicted** | |
|---|---|---|
| | **Pos** | **Neg** |
| **Pos** | 90 | 0 |
| **Neg** | 9 | 1 |
| **Total** | $90 + 10 = 100$ | |

$$\text{Precision} = 90/99 \qquad \text{Recall} = 90/90$$

$$F_1 = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} = (2 \cdot 90)/(2 \cdot 90 + 9 + 0) = 0.95$$

All great, except that the classifier sucks on the negative cases.

If you are concerned with the negative cases, swap the classes and compute another set of metrics.

# $F_1$ Score

- $F_1$ is often used as a summary score for binary classifiers instead of Accuracy.

  Works better with imbalanced classes.

- Criticised for giving Precision and Recall the same importance.
- Is not symmetric, ignores true negatives, i.e., is misleading for some cases of imbalanced classes.
- *Fowlkes-Mallows index* is a geometric mean of Precision and Recall (used in clustering).

  The geometric mean is between the arithmetic and harmonic mean. For example, the geometric mean of 2/3 and 1/3 is (approx) 0.4714.

# More Derived Metrics

| | |
|---|---|
| Positive predictive value (PPV), precision $= \frac{TP}{PP} = 1 - FDR$ | False omission rate (FOR) $= \frac{FN}{PN} = 1 - NPV$ |
| False discovery rate (FDR) $= \frac{FP}{PP} = 1 - PPV$ | Negative predictive value (NPV) $= \frac{TN}{PN} = 1 - FOR$ |

You can see that the negative predictive value becomes the Precision when we swap the classes (and vice versa).

# More Derived Metrics

| | |
|---|---|
| True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power $= \frac{TP}{P} = 1 - FNR$ | False negative rate (FNR), miss rate $= \frac{FN}{P} = 1 - TPR$ |
| False positive rate (FPR), probability of false alarm, fall-out $= \frac{FP}{N} = 1 - TNR$ | True negative rate (TNR), specificity (SPC), selectivity $= \frac{TN}{N} = 1 - FPR$ |

Note that *specificity* becomes Recall when we swap the classes (and vice versa).

For example, medical doctors communicate in terms of *sensitivity* and *specificity*.

| Actual condition | Predicted condition | |
|---|---|---|
| | **Cancer** | **Non-cancer** |
| **Cancer** | TP = 6 | FN = 2 |
| **Non-cancer** | FP = 1 | TN = 3 |
| **Total** | 8 + 4 = 12 | |

$$\text{TPR} = \text{Sensitivity} = \text{Recall} = \text{TP}/\text{P} = 6/8$$

How often is positive predicted positive?

$$\text{TNR} = \text{Specificity} = \text{TN}/\text{N} = 3/4$$

How often is negative predicted negative?

$$\text{FPR} = \text{Prob. of false alarm} = \text{FP}/\text{N} = 1/4$$

How often is negative predicted positive?

$$\text{FNR} = \text{Miss rate} = \text{FN}/\text{P} = 2/8$$

How often is positive predicted negative?

# Evaluating Multi-class Classifiers

# Classification Into Multiple Classes

Assume classification into classes from a finite set $C$.

Consider a classification dataset:

$$\{(\vec{x}_k, c_k) \mid k = 1, \ldots, p\}$$

Here $\vec{x}_k$ is a vector of attributes/features and $c_k \in C$ for all $k$.

Consider a sequence of predictions generated by a classifier:

$$h_1, \ldots, h_p \in C$$

Here each $h_k$ has been predicted for the $k$-the example $(\vec{x}_k, c_k)$.

How good are the predictions $h_1, \ldots, h_p$ w.r.t. $c_1, \ldots, c_p$?

There are many possible metrics ...

Consider an arbitrary (finite) number of classes in $C$.

# Confusion Matrix

Assume that $C = \{1, \ldots, m\}$.

Now, given two classes $i, j \in C$ we denote by $M_{ij}$ the number of samples of class $i$ classified into the class $j$.

Formally,

$$M_{ij} = |\{k \mid c_k = i \wedge h_k = j\}|$$

| **Actual** | **Predicted** | | | | |
|---|---|---|---|---|---|
| | $1$ | $\cdots$ | $j$ | $\cdots$ | $m$ |
| $1$ | $M_{11}$ | $\cdots$ | $M_{1j}$ | $\cdots$ | $M_{1m}$ |
| $\vdots$ | $\vdots$ | | $\vdots$ | | $\vdots$ |
| $i$ | $M_{i1}$ | $\cdots$ | $M_{ij}$ | $\cdots$ | $M_{im}$ |
| $\vdots$ | $\vdots$ | | $\vdots$ | | $\vdots$ |
| $m$ | $M_{m1}$ | $\cdots$ | $M_{mj}$ | $\cdots$ | $M_{mm}$ |

# Example

| **Actual** | **Predicted** |
|---|---|
| big | big |
| big | big |
| small | big |
| medium | medium |
| big | small |
| big | big |
| small | small |
| small | small |
| medium | medium |
| medium | small |
| small | small |
| big | big |
| medium | small |
| small | medium |
| big | big |

# Example

| Actual | Predicted |
|---|---|
| big | big |
| big | big |
| small | big |
| medium | medium |
| big | small |
| big | big |
| small | small |
| small | small |
| medium | medium |
| medium | small |
| small | small |
| big | big |
| medium | small |
| small | medium |
| big | big |

| Actual | Predicted | | |
|---|---|---|---|
| | **big** | **medium** | **small** |
| **big** | 5 | 0 | 1 |
| **medium** | 0 | 2 | 2 |
| **small** | 1 | 1 | 3 |

Note that the diagonal counts the correctly classified samples.

The off-diagonal elements correspond to misclassified samples.

# Metrics

We can easily generalize Accuracy, Precision, Recall, and $F_1$-score from the binary classification to multiple classes.

Notation

- $M_{i\bullet} = \sum_{j=1}^{m} M_{ij}$
- $M_{\bullet j} = \sum_{i=1}^{m} M_{ij}$
- $M_{\bullet\bullet} = \sum_{i=1}^{m} \sum_{j=1}^{m} M_{ij}$

Now, the metrics:

$$\text{Accuracy} = \frac{\sum_{k=1}^{m} M_{kk}}{M_{\bullet\bullet}}$$

For a given class $i \in C$:

$$\text{Precision}[i] = \frac{M_{ii}}{M_{\bullet i}} \qquad \text{Recall}[i] = \frac{M_{ii}}{M_{i\bullet}}$$

$$F_1[i] = \frac{2 * \text{Precision}[i] * \text{Recall}[i]}{\text{Precision}[i] + \text{Recall}[i]}$$

Note that Precision, Recall, and $F_1$ can be defined only for a given class!

# Example

| Actual | Predicted | | |
|---|---|---|---|
| | **big** | **medium** | **small** |
| **big** | 5 | 0 | 1 |
| **medium** | 0 | 2 | 2 |
| **small** | 1 | 1 | 3 |

Compute the metrics.

# Example

Accuracy $= (5+2+3)/15 = 0.66$

Precision[big] $= 5/6$

Precision[medium] $= 2/3$

Precision[small] $= 3/6$

Recall[big] $= 5/6$

Recall[medium] $= 2/4$

Recall[small] $= 3/5$

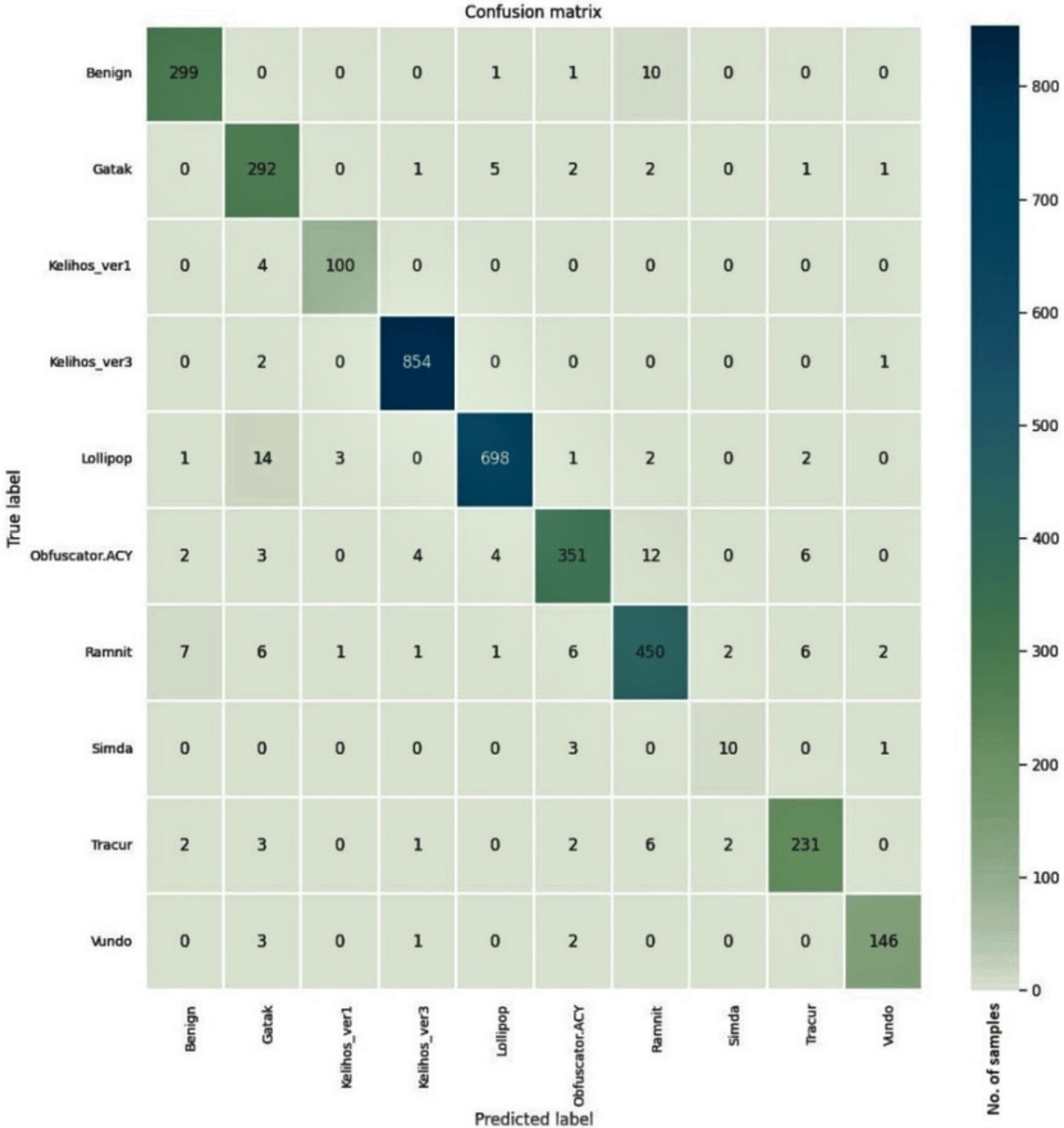
$$F_1[\text{big}] = \frac{2 * (5/6) * (5/6)}{(5/6) + (5/6)} = 5/6 = 0.83$$

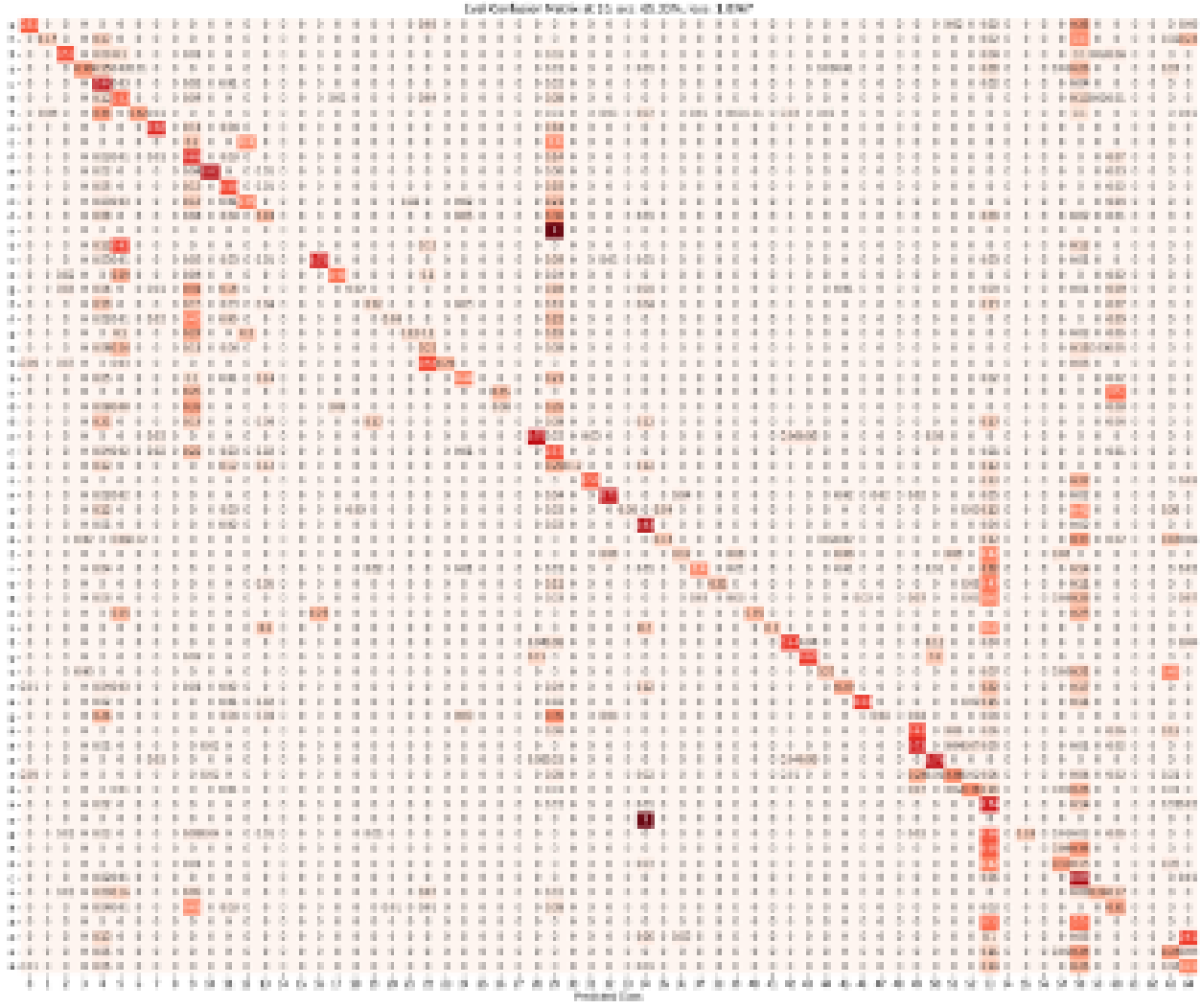
$F_1$[medium] $= 0.57$

$F_1$[medium] $= 0.54$

| **Actual** | **Predicted** | | |
|---|---|---|---|
| | **big** | **medium** | **small** |
| **big** | 5 | 0 | 1 |
| **medium** | 0 | 2 | 2 |
| **small** | 1 | 1 | 3 |

How do you get a single number out of these? Average Precision, Recall, and $F_1$ are usually computed, but one needs to be careful about the variance.


Confusion matrix
Benign 299 0 0 0 1 1 10 0 0 0
Gatak 0 292 0 1 5 2 2 0 1 1
Kelihos_ver1 0 4 100 0 0 0 0 0 0 0
Kelihos_ver3 0 2 0 854 0 0 0 0 0 1
Lollipop 1 14 3 0 698 1 2 0 2 0
Obfuscator.ACY 2 3 0 4 4 351 12 0 6 0
Ramnit 7 6 1 1 1 6 450 2 6 2
Simda 0 0 0 0 0 3 0 10 0 1
Tracur 2 3 0 1 0 2 6 2 231 0
Vundo 0 3 0 1 0 2 0 0 0 146
True label
Benign
Gatak
Kelihos_ver1
Kelihos_ver3
Lollipop
Obfuscator.ACY
Ramnit
Simda
Tracur
Vundo
Predicted label
800
700
600
500
400
300
200
100
0
No. of samples



Machine learning/data mining is needed to understand the matrix.

# Probabilistic Classifier Evaluation

# Binary Probabilistic Classifier

Assume binary classification into two classes $\{0, 1\}$.

Consider a classification dataset:

$$\{(\vec{x}_k, c_k) \mid k = 1, \ldots, p\}$$

Here $\vec{x}_k$ is a vector of attributes/features and $c_k \in C$ for all $k$.

Consider a sequence of predictions generated by a classifier.
Now the classifier returns *probability of class* 1 for a given input:

$$h_1, \ldots, h_p \in [0, 1]$$

Here each $h_k$ has been predicted for the $k$-the example $(\vec{x}_k, c_k)$.

How to interpret the predictions $h_1, \ldots, h_p$?

How good are the predictions $h_1, \ldots, h_p$ w.r.t. $c_1, \ldots, c_p$?

# Probabilistic Classifier

Let us fix predictions $h_1, \ldots, h_p$.

Given a threshold $T \in [0, 1]$ we define

$$h_k^T = \begin{cases} 1 & \text{if } h_k \geq T \\ 0 & \text{if } h_k < T \end{cases}$$

For every $T$ we can compute all the metrics (Precision, Recall, etc.)

Given a metric MET and a threshold $T$, we denote by MET$[T]$ the metric MET evaluated on $h_1^T, \ldots, h_p^T$.

We obtain

$$\text{TP}[T] = |\{k \mid h_k^T = 1 \wedge c_k = 1\}|$$

and
$\text{TN}[T], \text{FP}[T], \text{FN}[T], \text{Accuracy}[T], \text{Precision}[T], \text{Recall}[T], F_1[T], \ldots$

However, all metrics are now functions of the threshold $T$.

# Thresholded Classifier Metrics

| **Index** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Actual** | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| **Predicted** | .98 | .95 | .9 | .86 | .66 | .48 | .43 | .42 | .36 | .15 | .1 | .05 |
| T=0.5 | TP | TP | TP | TP | TP | TN | TN | FN | FN | TN | TN | TN |
| T=0.42 | TP | TP | TP | TP | TP | FP | FP | TP | FN | TN | TN | TN |
| T=0.1 | TP | TP | TP | TP | TP | FP | FP | TP | TP | FP | FP | TN |

For example, consider $T = 0.42$, then

$$\text{TP}[T] = 6 \quad \text{FP}[T] = 2 \quad \text{FN}[T] = 1 \quad \text{TN}[T] = 3$$

$$\text{Accuracy}[T] = \frac{3+6}{12} \quad \text{Precision}[T] = \frac{6}{6+2} \quad \text{Recall}[T] = \frac{6}{6+1}$$

$$F_1[T] = \frac{2 \cdot 6/8 \cdot 6/7}{6/8 + 6/7} = 0.8$$

# Receiver Operating Characteristic (ROC)

Consider two metrics for a given $T$:

$$\mathrm{TPR}[T] = \frac{\mathrm{TP}[\mathrm{T}]}{\mathrm{P}[T]} \qquad \text{(True Positive Rate)}$$

$$\mathrm{FPR}[T] = \frac{\mathrm{FP}[T]}{\mathrm{N}[T]} \qquad \text{(False Positive Rate)}$$

*ROC curve* is then a function $\mathrm{ROC} : [0,1] \to [0,1]^2$ defined by

$$\mathrm{ROC}(T) = (\mathrm{TPR}[T], \mathrm{FPR}[T])$$

Observe that

$$\mathrm{ROC}(0) = (1,1)$$

Because the classifier with $T = 0$ simply classifies everything as positive, i.e., into the class 1.

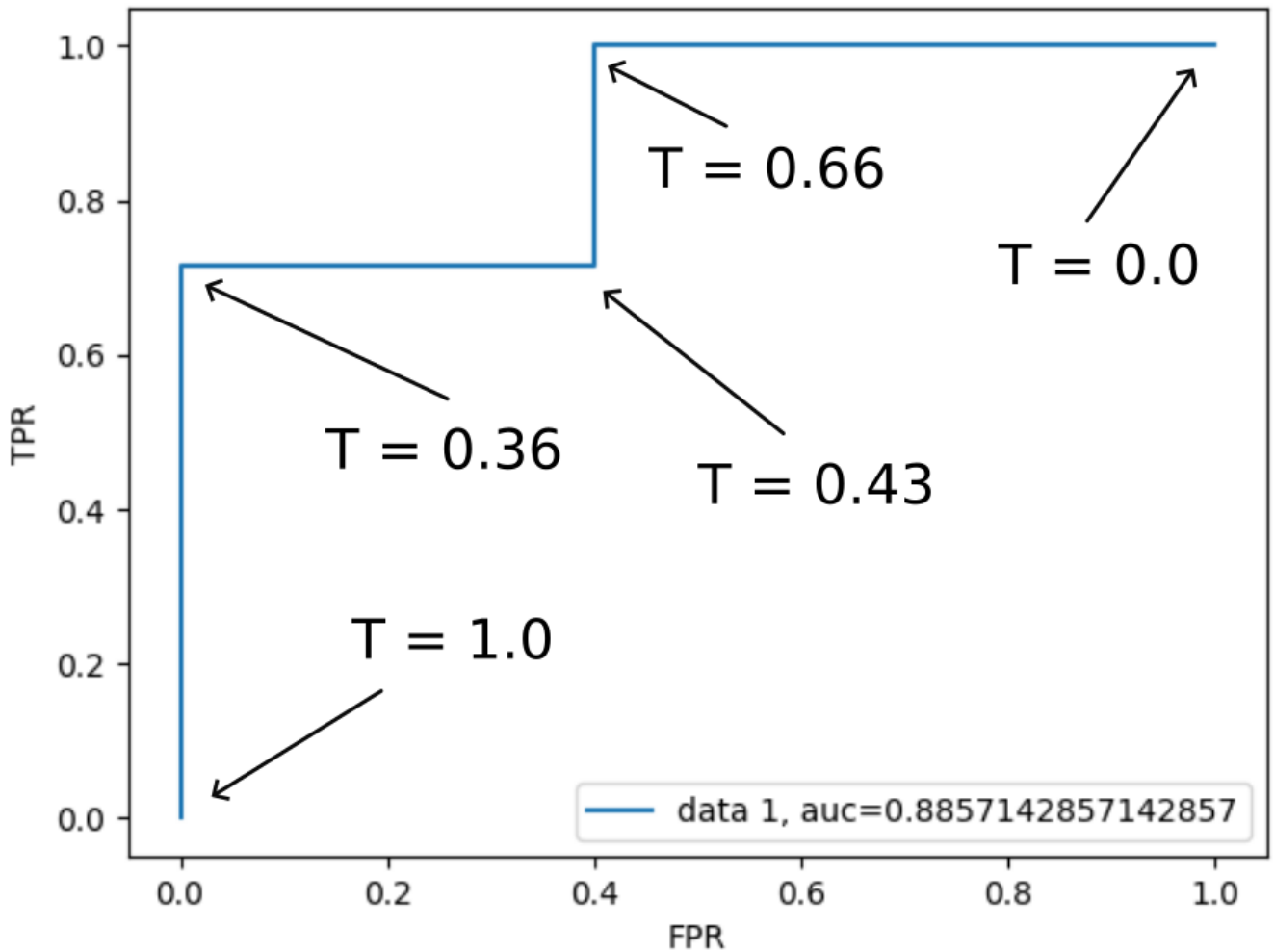
Both $\mathrm{TPR}[T]$ and $\mathrm{FPR}[T]$ are non-increasing in $T$.

| **Index** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Actual** | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| **Predicted** | .98 | .95 | .9 | .86 | .66 | .48 | .43 | .42 | .36 | .15 | .1 | .05 |

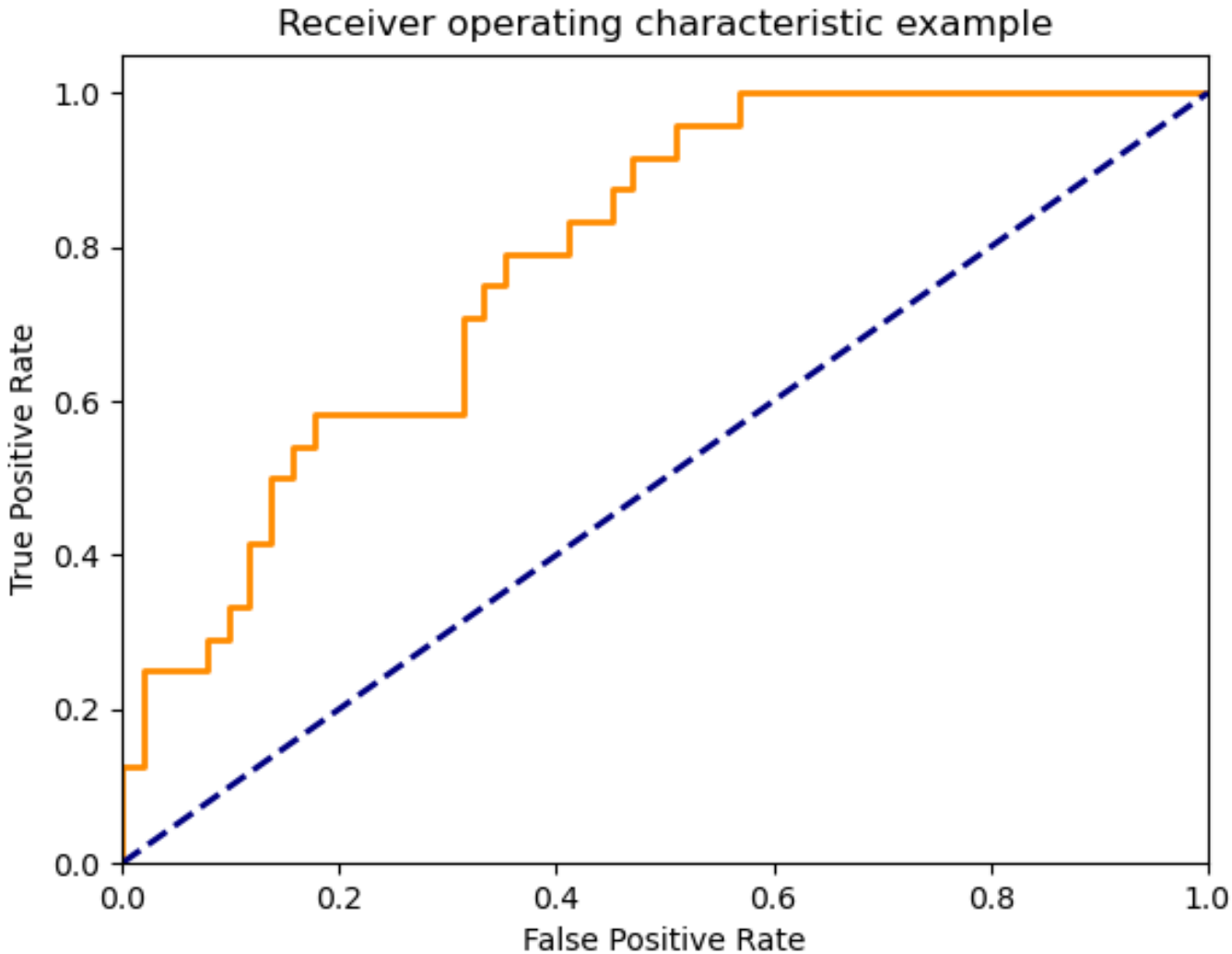
- $0.00 \leq T \leq 0.05$: TPR $= 1$ and FPR $= 1$
- $0.05 < T \leq 0.10$: TPR $= 1$ and FPR $= 4/5$
- $0.10 < T \leq 0.15$: TPR $= 1$ and FPR $= 3/5$
- $0.15 < T \leq 0.36$: TPR $= 1$ and FPR $= 2/5$
- $0.36 < T \leq 0.42$: TPR $= 6/7$ and FPR $= 2/5$
- $0.42 < T \leq 0.43$: TPR $= 5/7$ and FPR $= 2/5$
- $0.43 < T \leq 0.48$: TPR $= 5/7$ and FPR $= 1/5$
- $0.48 < T \leq 0.66$: TPR $= 5/7$ and FPR $= 0$
- $0.66 < T \leq 0.86$: TPR $= 4/7$ and FPR $= 0$
- $0.86 < T \leq 0.90$: TPR $= 3/7$ and FPR $= 0$
- $0.90 < T \leq 0.95$: TPR $= 2/7$ and FPR $= 0$
- $0.95 < T \leq 0.98$: TPR $= 1/7$ and FPR $= 0$
- $0.98 < T \leq 1.00$: TPR $= 0$ and FPR $= 0$

# ROC

| **Index** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Actual** | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| **Predicted** | .98 | .95 | .9 | .86 | .66 | .48 | .43 | .42 | .36 | .15 | .1 | .05 |

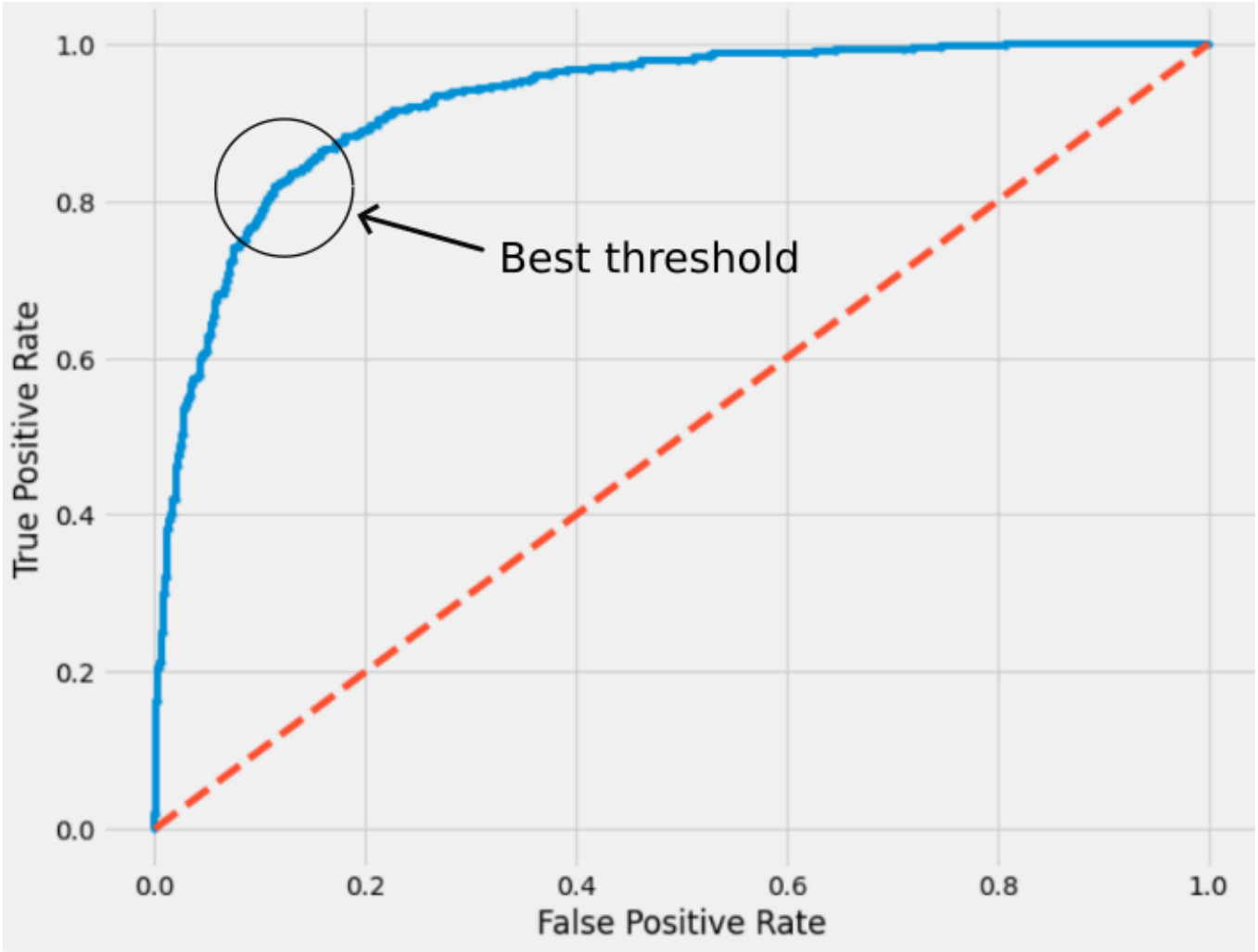


# Iris Dataset - A Classifier



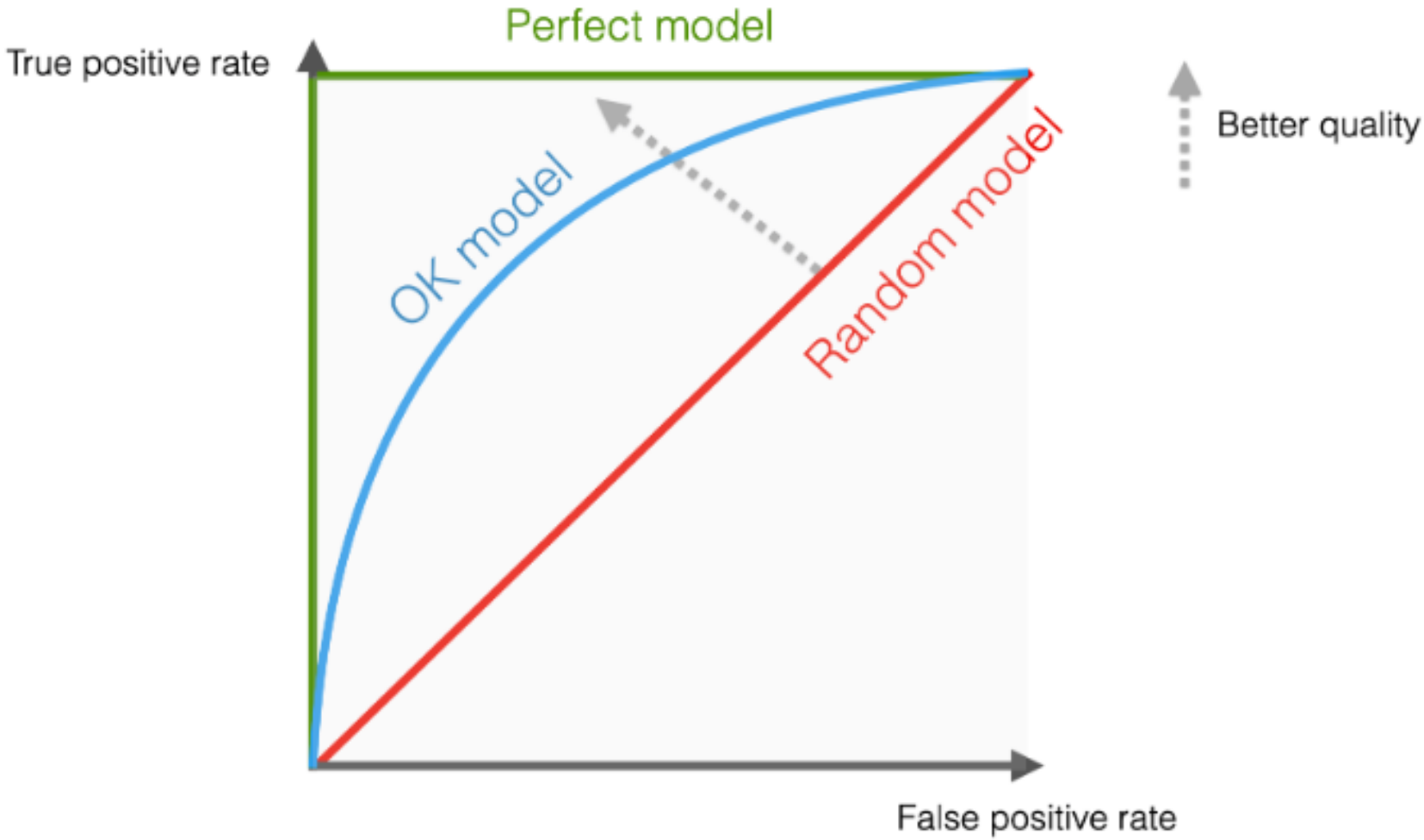

Example from the scikit-learn manual - SVM classifier trained in Iris

# Using ROC and Threshold




Search for the best threshold at the elbow of the ROC curve.

# ROC - Explanation

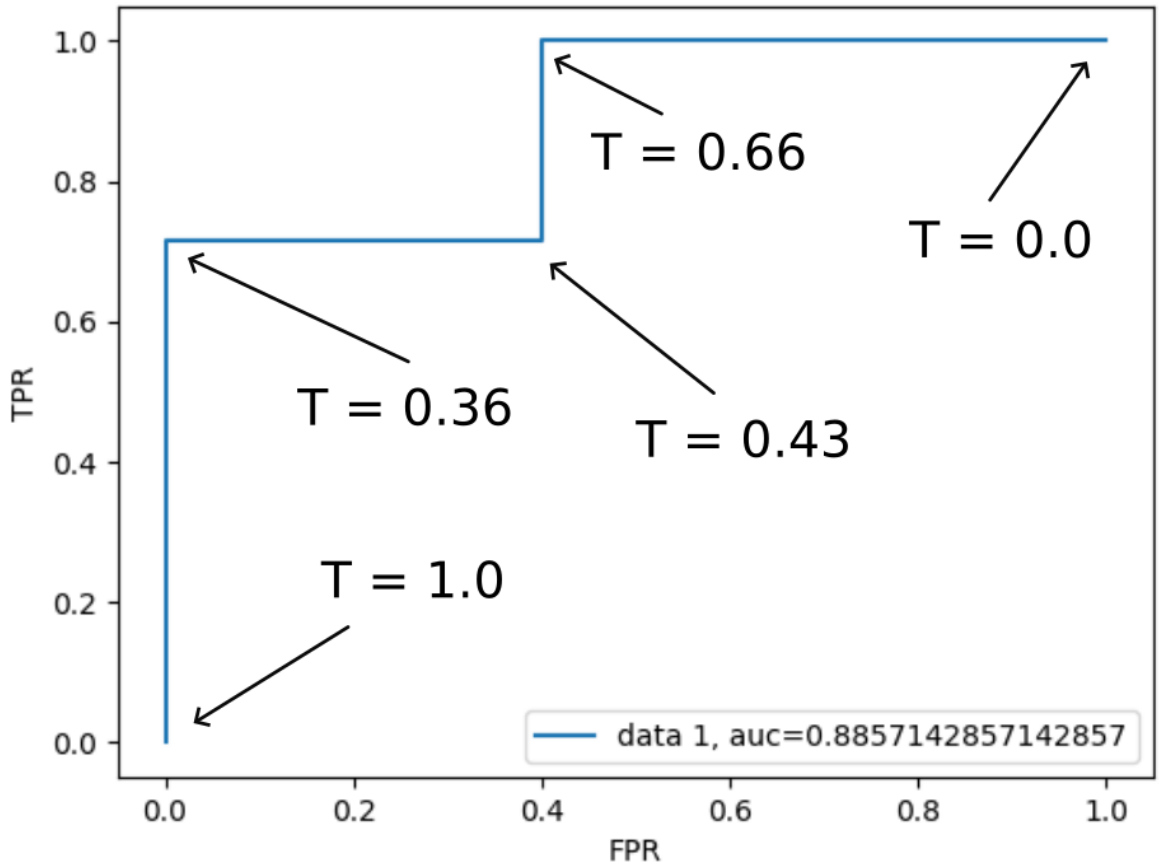



The larger the *area under the ROC curve (ROC-AUC)*, the better.

ROC-AUC ranges from 0 to 1. ROC-AUC $\approx 0.5$ indicates random guessing.

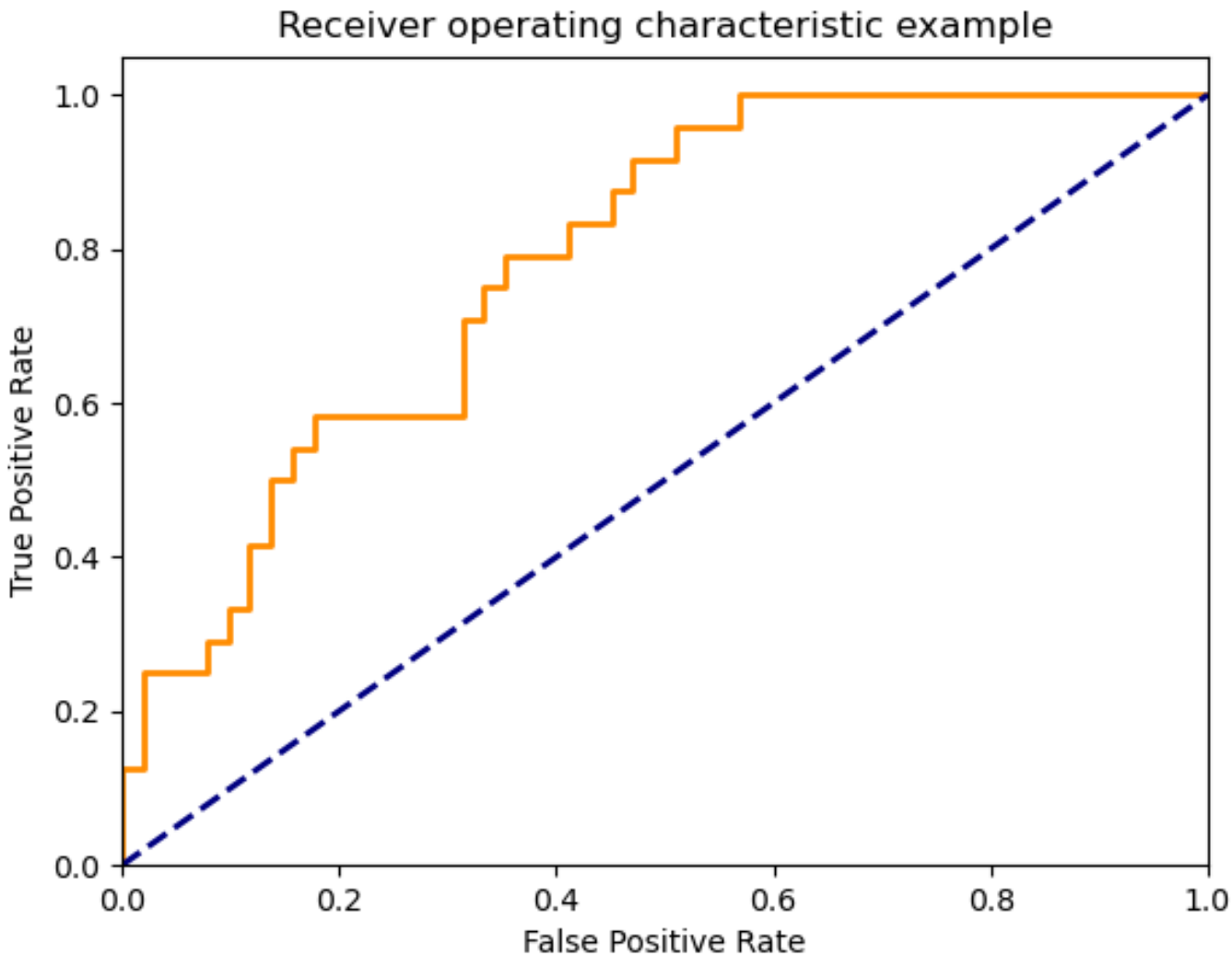
# ROC-AUC

| **Index** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Actual** | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| **Predicted** | .98 | .95 | .9 | .86 | .66 | .48 | .43 | .42 | .36 | .15 | .1 | .05 |




ROC-AUC $= 0.8857$

# Iris - ROC-AUC




ROC-AUC $= 0.79$

# ROC-AUC - Probabilistic Interpretation

How is the ROC-AUC connected with the samples?

Consider our cancer detection example:

| **Index** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Actual** | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| **Predicted** | .98 | .95 | .9 | .86 | .66 | .48 | .43 | .42 | .36 | .15 | .1 | .05 |

AUC has a probabilistic explanation:

Consider the following experiment:

- Choose randomly a patient $i$ from positive patients

  Each positive patient has the same probability of being chosen.
- Choose randomly a patient $j$ from negative patients

  Each negative patient has the same probability of being chosen.
- Check if $h_i > h_j$.

The ROC-AUC is the probability of succeeding in the $h_i > h_j$ test.

# Summary

We have discussed various metrics that can be used to evaluate the quality of a classifier.

The metrics summarize the results of evaluation on a given dataset.

We have discussed metrics for evaluating

- binary classifiers,
  Accuracy, Precision, Recall, $F_1$, and few more
- multi-class classifiers,
  Accuracy, Precision, Recall, $F_1$
- probabilistic classifiers,
  parametrized metrics, ROC-AUC

There are still several questions unanswered:

- When to use the metrics.
- How to estimate the influence of sampling the dataset.

# Use of Evaluation Metrics

In our case, the following scenarios are typical:

- **Final test**: Evaluate the model on the test set (separated at the beginning of training) and then compute the metrics. May inform the user about the quality of the model.
- **Validation**: Evaluate models on a separate validation set and use the metrics to compare models.
  There are (at least) two scenarios in which this happens:
  - Hyperparameter fine-tuning.
  - Comparison of different models (e.g., KNN and decision trees).

Keep in mind that the metrics are artificial, and the results of the model are roughly summarized.

It would be best if you always strived to test the proper functionality of your model in as natural conditions as possible.

For example, a model for medical diagnosis should be evaluated by medical doctors who may observe many features of its behavior that are difficult to express quantitatively.

# How to Estimate Significance

Machine learning models are typically trained on (pseudo) random samples of data objects.

For example, a set of patients treated by the concrete hospital.

However, the purpose of testing/evaluation is to get information about the whole population (i.e., all possible patients).

How do we estimate how much specific properties of the given sample influence our model?

This is a challenging question; methods of inferential statistics are needed to get the answer.

We will consider these issues in some later lecture. Concretely,

- *Bias-variance* tradeoff
- *Statistical tests* for testing
  - significance of the metrics values,
  - paired t-tests for comparing models.

# How to Compare Classifiers

Let us consider two classifiers. How do you compare them?

Accuracies and $F_1$ scores can be compared easily (they are just numbers).

How to compare $(\text{Precision}_1, \text{Recall}_1)$ of the fist classifier with $(\text{Precision}_2, \text{Recall}_2)$ of the second classifier?

*Thresholding*

- Introduce a threshold $0 \leq t \leq 1$
- Demand, one of the two metrics (typically the Recall), to be at least $t$. That is

$$\text{Recall}_1 \geq t \qquad \text{Recall}_2 \geq t$$

- Compare the values of the other metric numerically. In our case, decide whether

$$\text{Precision}_1 \geq \text{Precision}_2$$

(Still need to be concerned about the statistical significance.)

# Example

| Actual condition | Predicted condition | |
|---|---|---|
| | Canc. | Non-canc. |
| **Cancer** | 6 | 2 |
| **Non-canc.** | 1 | 3 |
| **Total** | $8 + 4 = 12$ | |

| Actual condition | Predicted condition | |
|---|---|---|
| | Canc. | Non-canc. |
| **Cancer** | 5 | 3 |
| **Non-canc.** | 0 | 4 |
| **Total** | $8 + 4 = 12$ | |

$$\text{Precision}_1 = \frac{6}{7} \qquad \text{Recall}_1 = \frac{6}{8}$$

$$\text{Precision}_2 = \frac{5}{5} = 1 \qquad \text{Recall}_2 = \frac{5}{8}$$

Consider a threshold $t$ on the Recall.

The second classifier is better if the threshold $t$ is $5/8$, then the second classifier is better.

If the threshold $t$ is $6/8$, then the second classifier is unacceptable.

# Numerical features

- Throughout this lecture we assume that all features are numerical, i.e., feature vectors belong to $\mathbb{R}^n$.
- Most non-numerical features can be conveniently transformed to numerical ones.

  **For example:**
  - Colors $\{blue, red, yellow\}$ can be represented by

    $$\{(1,0,0),(0,1,0),(0,0,1)\}$$

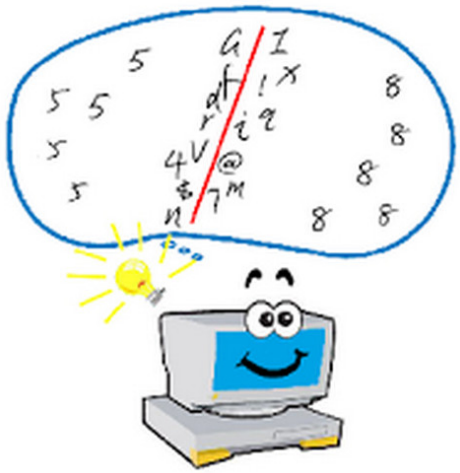
    (one-hot encoding)
  - Words can be embedded into vector spaces by various means (word2vec etc.)
  - A black-and-white picture of $x \times y$ pixels can be encoded as a vector of $xy$ numbers that capture the shades of gray of the pixels.
    (Even though this is not the best way of representing images.)

# Basic Problems

We consider two basic problems:

- (Binary) classification

  **Our goal:** Classify inputs into two categories.

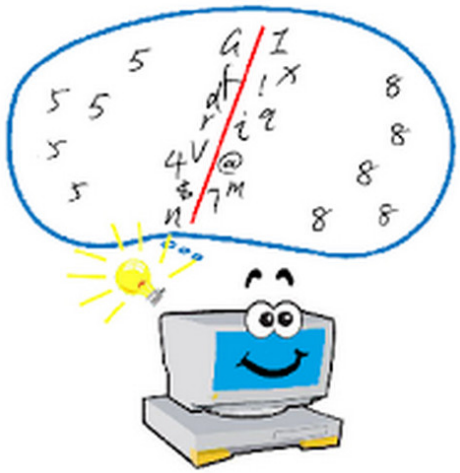


# Basic Problems

We consider two basic problems:

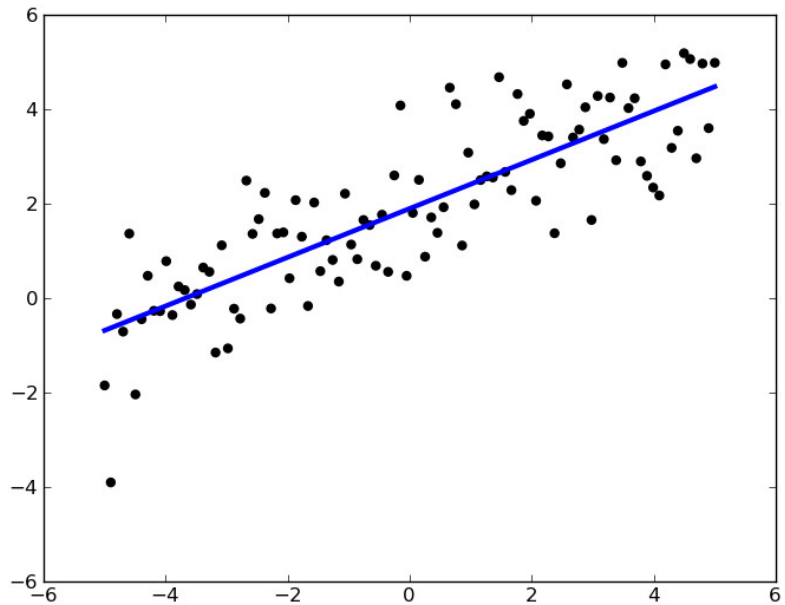
- (Binary) classification

  **Our goal:** Classify inputs into two categories.



- Regression

  **Our goal:** Find a (hypothesized) functional dependency in data.



# Linear Models

## Binary Classification

# Binary classification in $\mathbb{R}^n$

**Our goal:**

- Given a set $D$ of training examples of the form $(\vec{x}, c)$ where $\vec{x} \in \mathbb{R}^n$ and $c \in \{0, 1\}$,
- construct a model $h : \mathbb{R}^n \to \{0, 1\}$ that is consistent with $D$, i.e.,

$$h(\vec{x}) = c \text{ for all training examples } (\vec{x}, c) \in D$$

Comments:

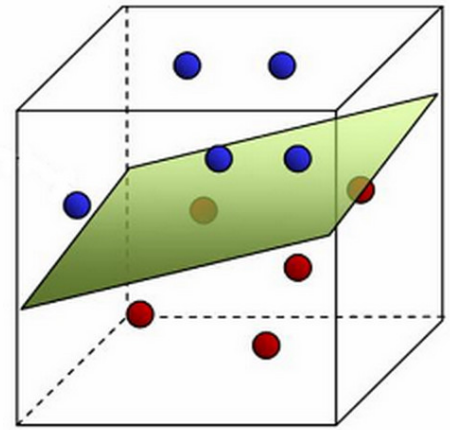
- In practice, we often do not strictly demand $h(\vec{x}) = c$ for all training examples $(\vec{x}, c) \in D$ (often it is impossible)
- We are more interested in good **generalization**, that is how well $h$ classifies new instances that do not belong to $D$.

  (Recall that we usually evaluate accuracy of the resulting hypothesized function $h$ on a test set.)

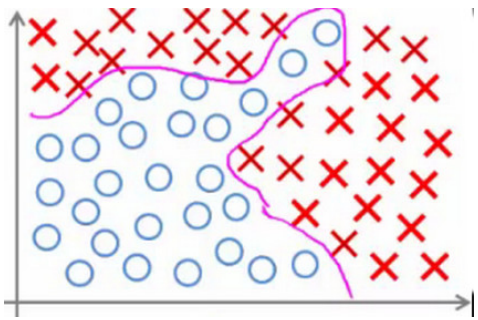
# Models

We consider two kinds of hypothesis spaces:

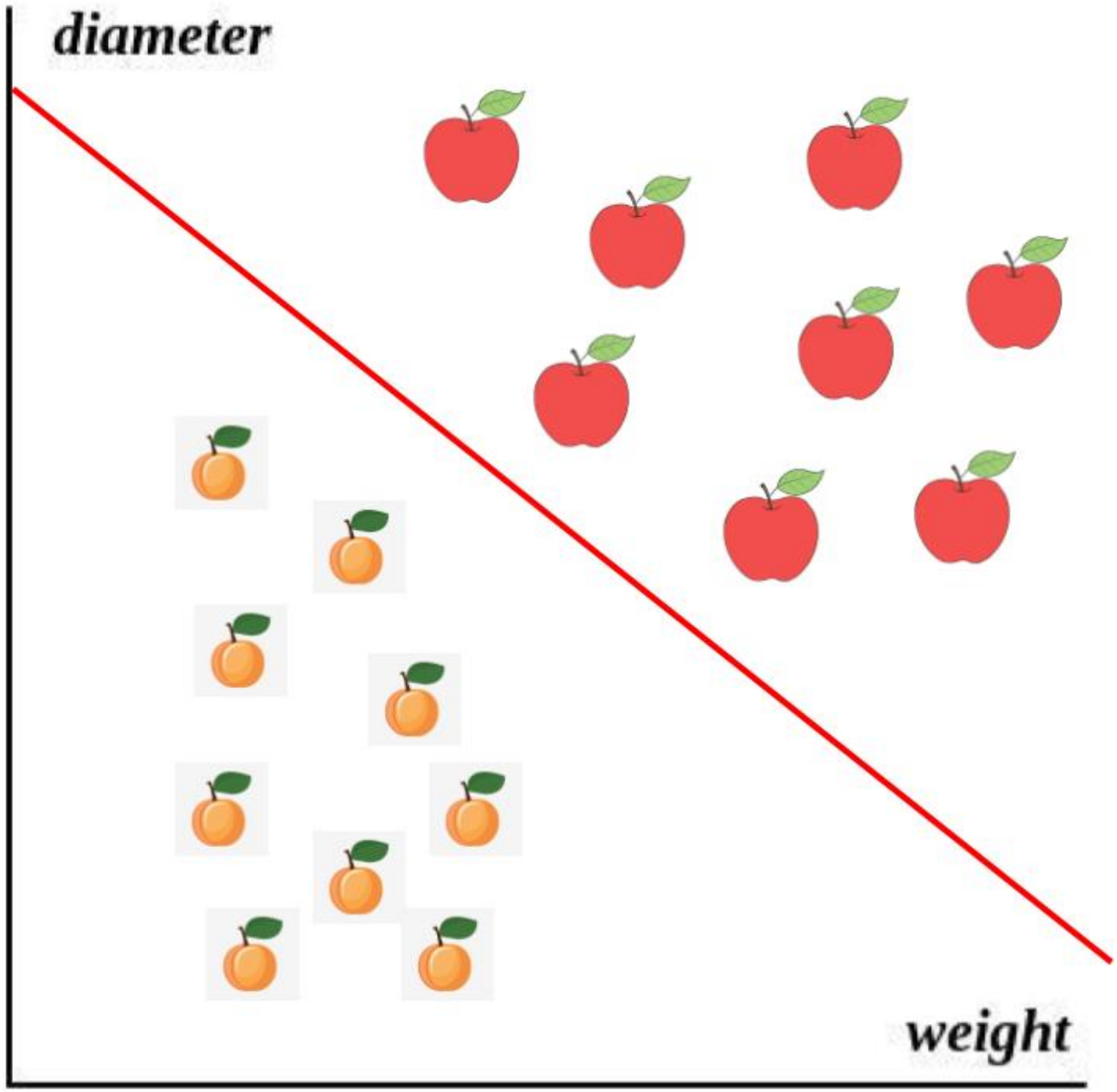
- Linear (affine) classifiers (this lecture)



- Non-linear classifiers (kernel SVM, neural networks) (later lectures)



# Linear Classifier – Example



# Length and Scalar Product of Vectors

- We consider vectors $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^m$.
- Euclidean metric on vectors: $||\vec{x}|| = \sqrt{\sum_{i=1}^{n} x_i^2}$
  The distance between two vectors (points) $\vec{x}, \vec{y}$ is $||\vec{x} - \vec{y}||$.
- *Scalar product* $\vec{x} \cdot \vec{y}$ of vectors $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$ defined by

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} x_i y_i$$

  - Recall that $\vec{x} \cdot \vec{y} = ||\vec{x}|| \, ||\vec{y}|| \cos\theta$ where $\theta$ is the angle between $\vec{x}$ and $\vec{y}$. That is $\vec{x} \cdot \vec{y}$ is the length of the projection of $\vec{y}$ on $\vec{x}$ multiplied by $||\vec{x}||$.
  - Note that $\vec{x} \cdot \vec{x} = ||\vec{x}||^2$

# Linear Classifier

A *linear classifier* $h[\vec{w}]$ is determined by a vector of *weights* $\vec{w} = (w_0, w_1, \ldots, w_n) \in \mathbb{R}^{n+1}$ as follows:

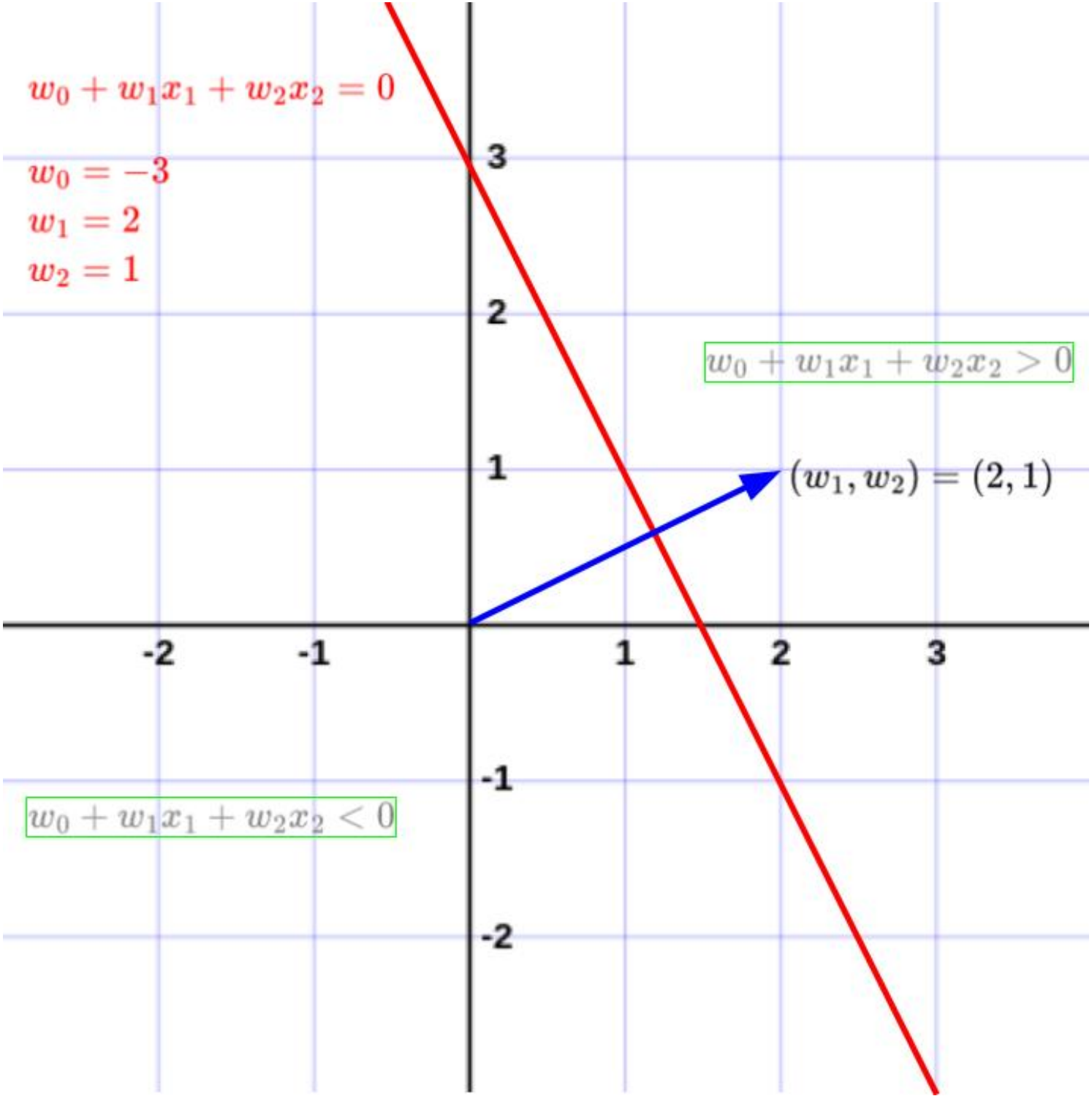
Given $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$,

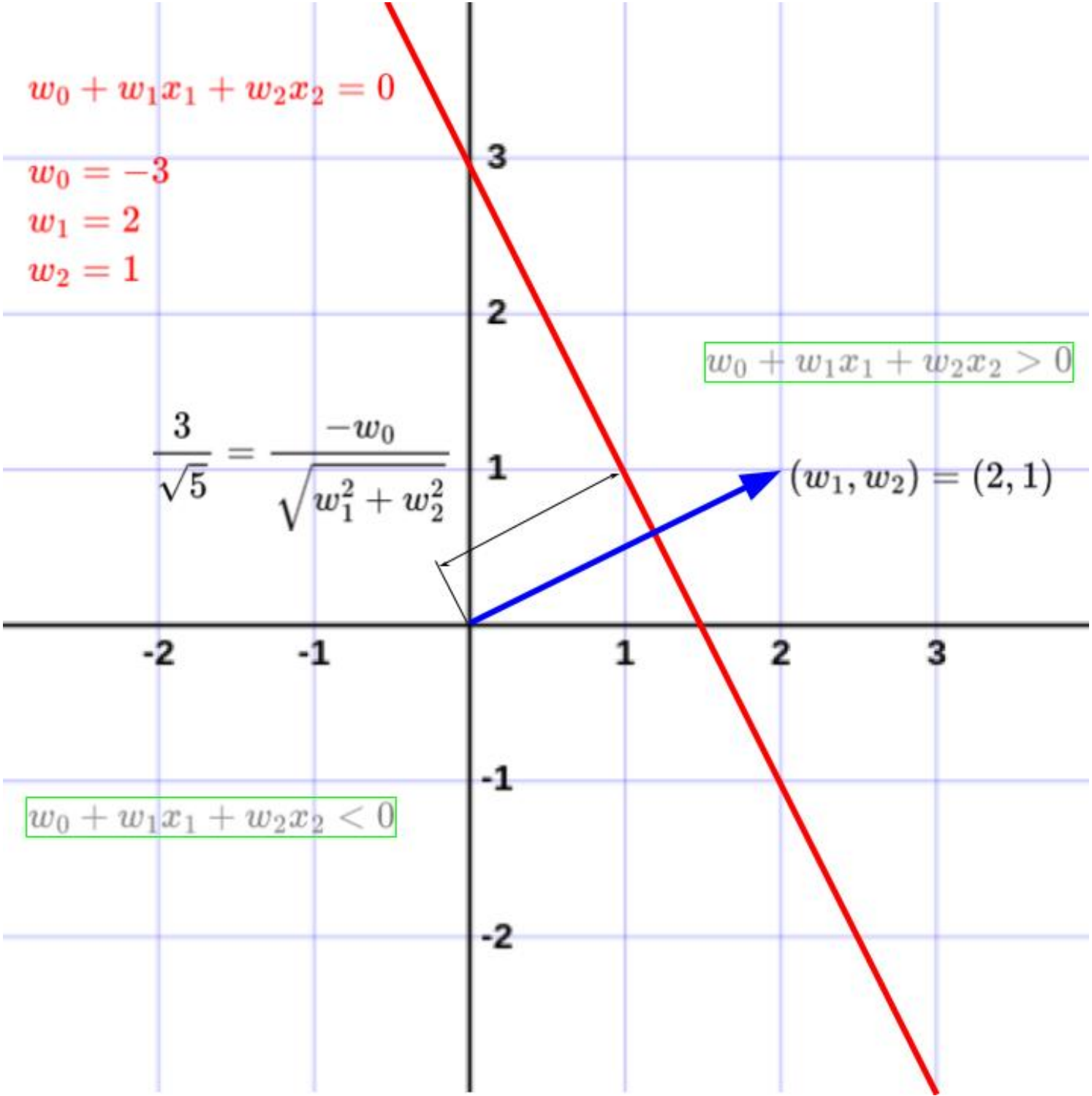
$$h[\vec{w}](\vec{x}) := \begin{cases} 1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i \geq 0 \\ 0 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i < 0 \end{cases}$$

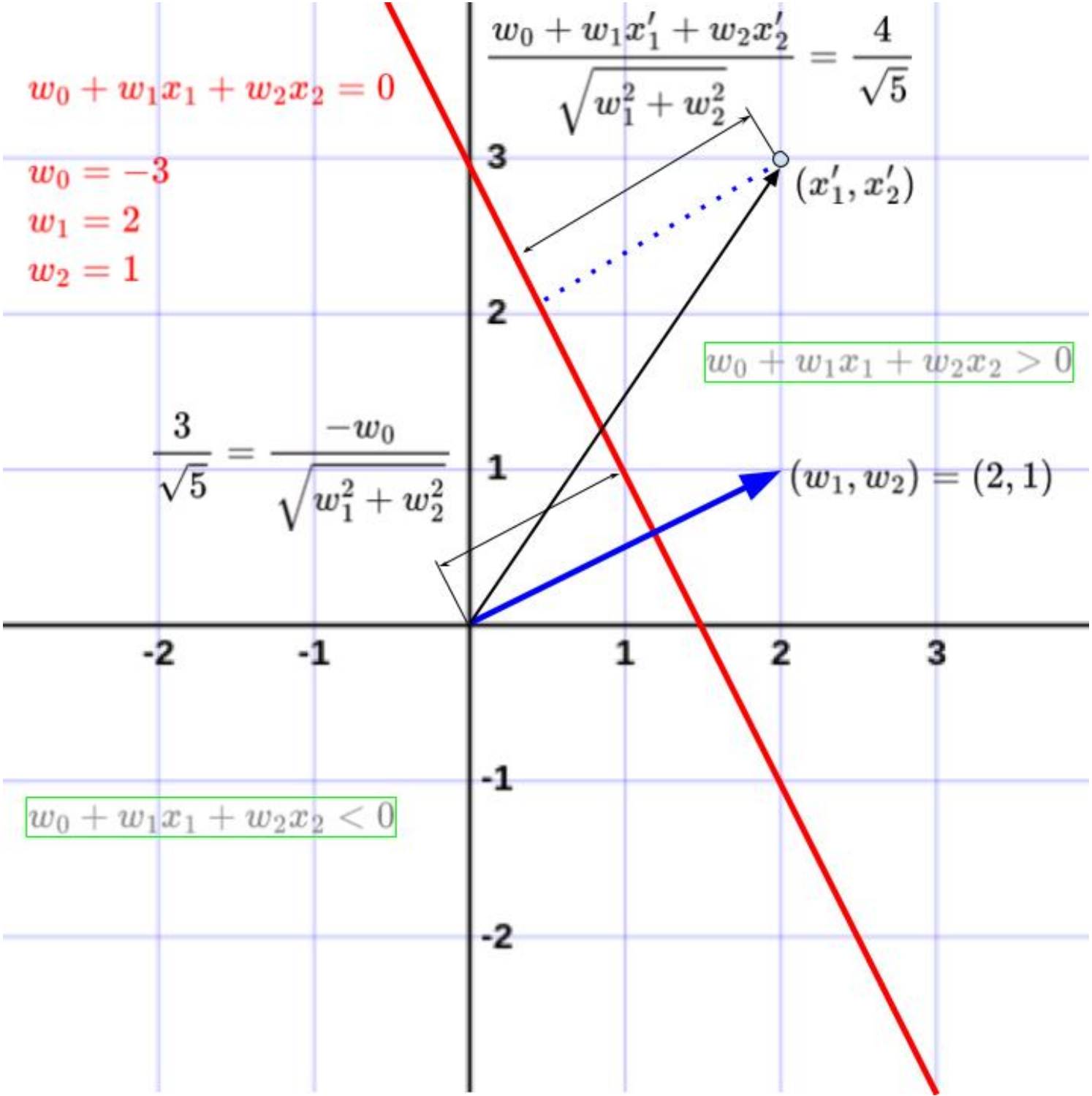
More succinctly:

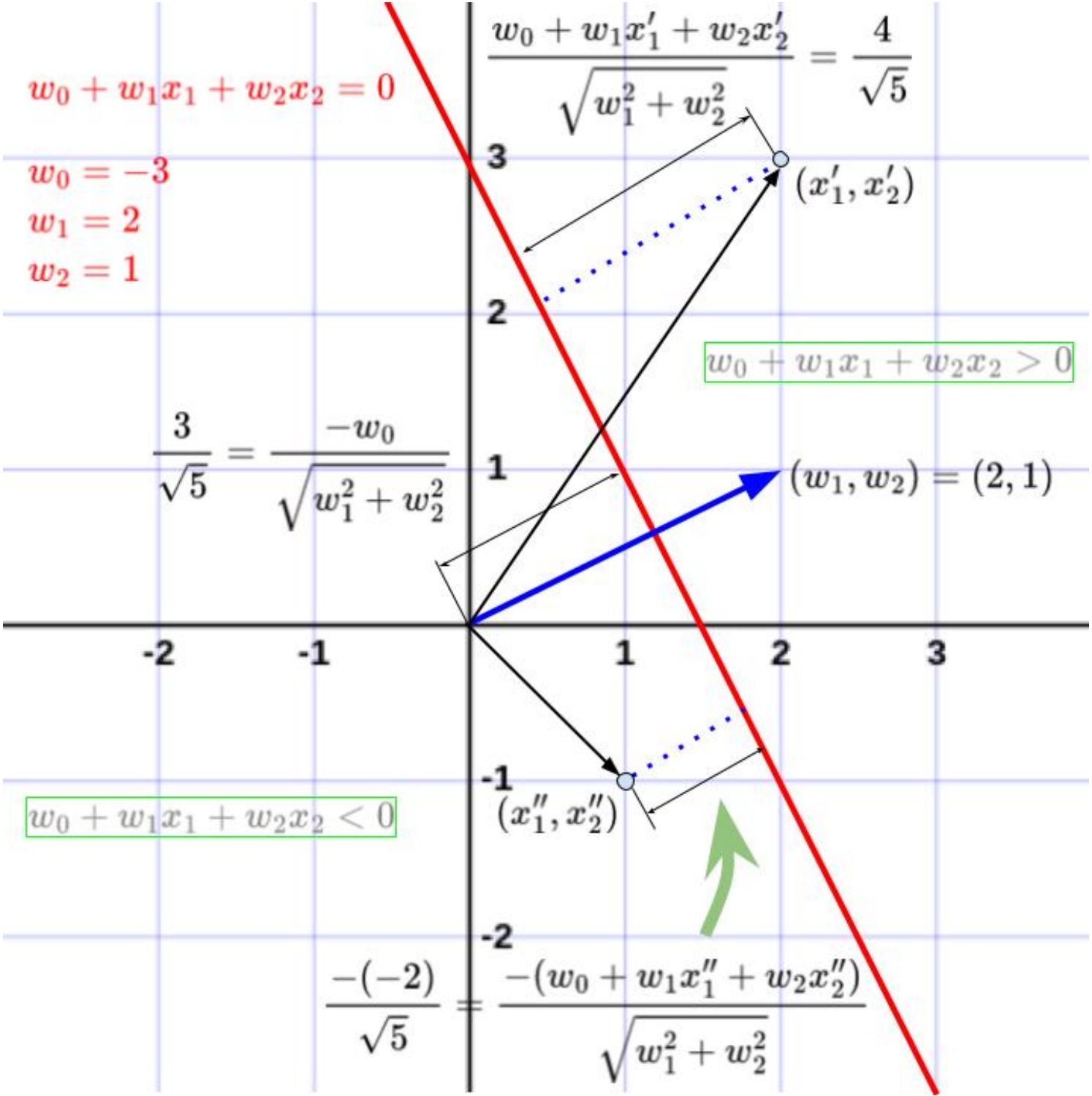
$$h(\vec{x}) = sgn\left(w_0 + \sum_{i=1}^{n} w_i \cdot x_i\right) \quad \text{where} \quad sgn(y) = \begin{cases} 1 & y \geq 0 \\ 0 & y < 0 \end{cases}$$

We define separating hyperplane determined by $\vec{w}$ as the set of all $\vec{x} \in \mathbb{R}^n$ satisfying $w_0 + \sum_{i=1}^{n} w_i \cdot x_i = 0$.

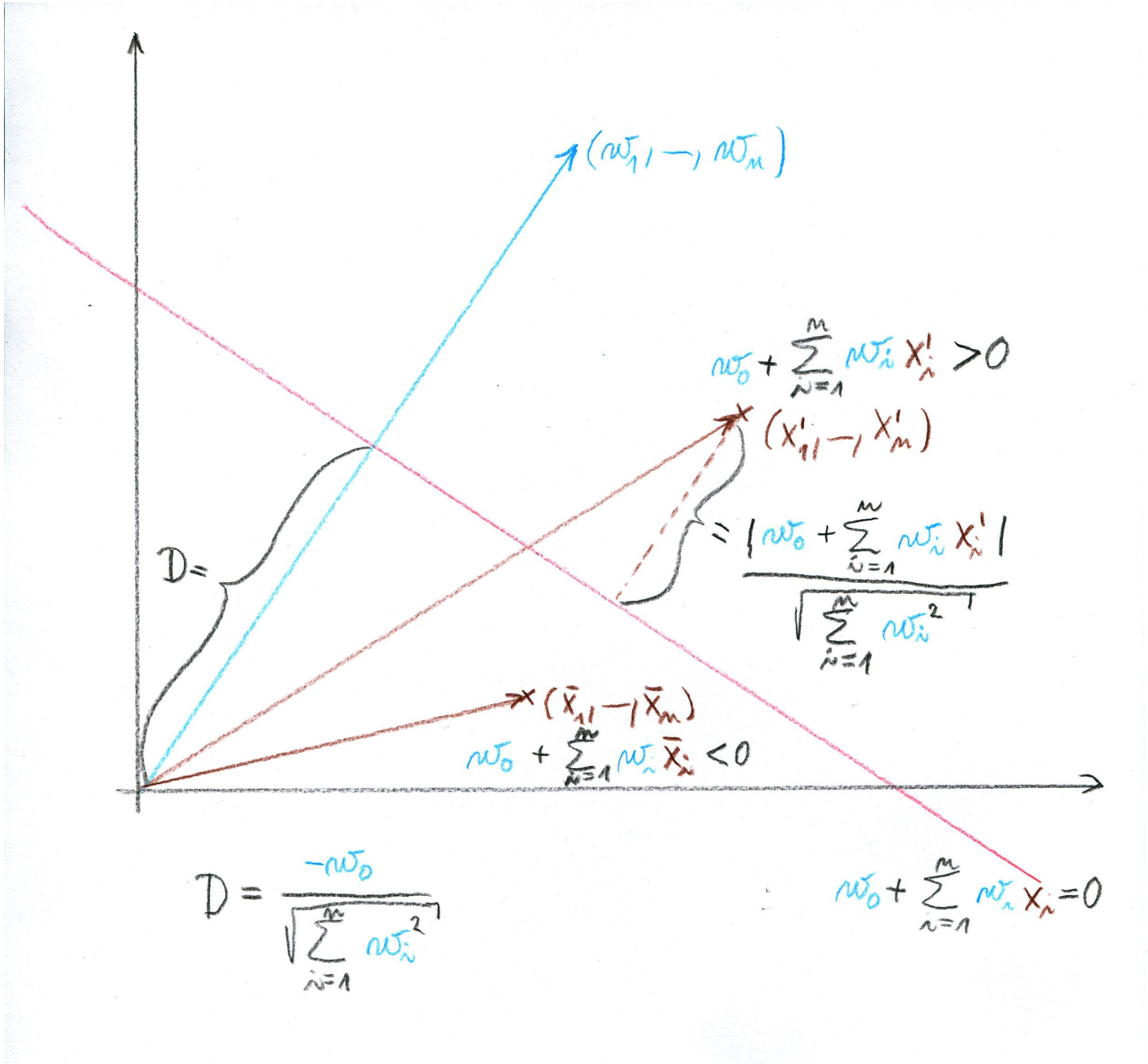

$w_0 + w_1 x_1 + w_2 x_2 = 0$
$w_0 = -3$
$w_1 = 2$
$w_2 = 1$
$w_0 + w_1 x_1 + w_2 x_2 > 0$
$(w_1, w_2) = (2, 1)$
$w_0 + w_1 x_1 + w_2 x_2 < 0$
3
2
1
-1
-2
-2
-1
1
2
3


$w_0 + w_1 x_1 + w_2 x_2 = 0$
$w_0 = -3$
$w_1 = 2$
$w_2 = 1$
$w_0 + w_1 x_1 + w_2 x_2 > 0$
$\frac{3}{\sqrt{5}} = \frac{-w_0}{\sqrt{w_1^2 + w_2^2}}$
$(w_1, w_2) = (2, 1)$
$w_0 + w_1 x_1 + w_2 x_2 < 0$
3
2
1
-1
-2
-2
-1
1
2
3


$w_0 + w_1 x_1 + w_2 x_2 = 0$
$w_0 = -3$
$w_1 = 2$
$w_2 = 1$
$\frac{w_0 + w_1 x'_1 + w_2 x'_2}{\sqrt{w_1^2 + w_2^2}} = \frac{4}{\sqrt{5}}$
$(x'_1, x'_2)$
$w_0 + w_1 x_1 + w_2 x_2 > 0$
$\frac{3}{\sqrt{5}} = \frac{-w_0}{\sqrt{w_1^2 + w_2^2}}$
$(w_1, w_2) = (2, 1)$
$w_0 + w_1 x_1 + w_2 x_2 < 0$
-2
-1
1
2
3
-1
-2


$w_0 + w_1 x_1 + w_2 x_2 = 0$
$w_0 = -3$
$w_1 = 2$
$w_2 = 1$
$\frac{w_0 + w_1 x_1' + w_2 x_2'}{\sqrt{w_1^2 + w_2^2}} = \frac{4}{\sqrt{5}}$
$(x_1', x_2')$
$w_0 + w_1 x_1 + w_2 x_2 > 0$
$\frac{3}{\sqrt{5}} = \frac{-w_0}{\sqrt{w_1^2 + w_2^2}}$
$(w_1, w_2) = (2, 1)$
$w_0 + w_1 x_1 + w_2 x_2 < 0$
$(x_1'', x_2'')$
$\frac{-(-2)}{\sqrt{5}} = \frac{-(w_0 + w_1 x_1'' + w_2 x_2'')}{\sqrt{w_1^2 + w_2^2}}$
3
2
1
-1
-2
-2
-1
1
2
3

# Linear Classifier – Geometry



# Linear Classifier – Notation

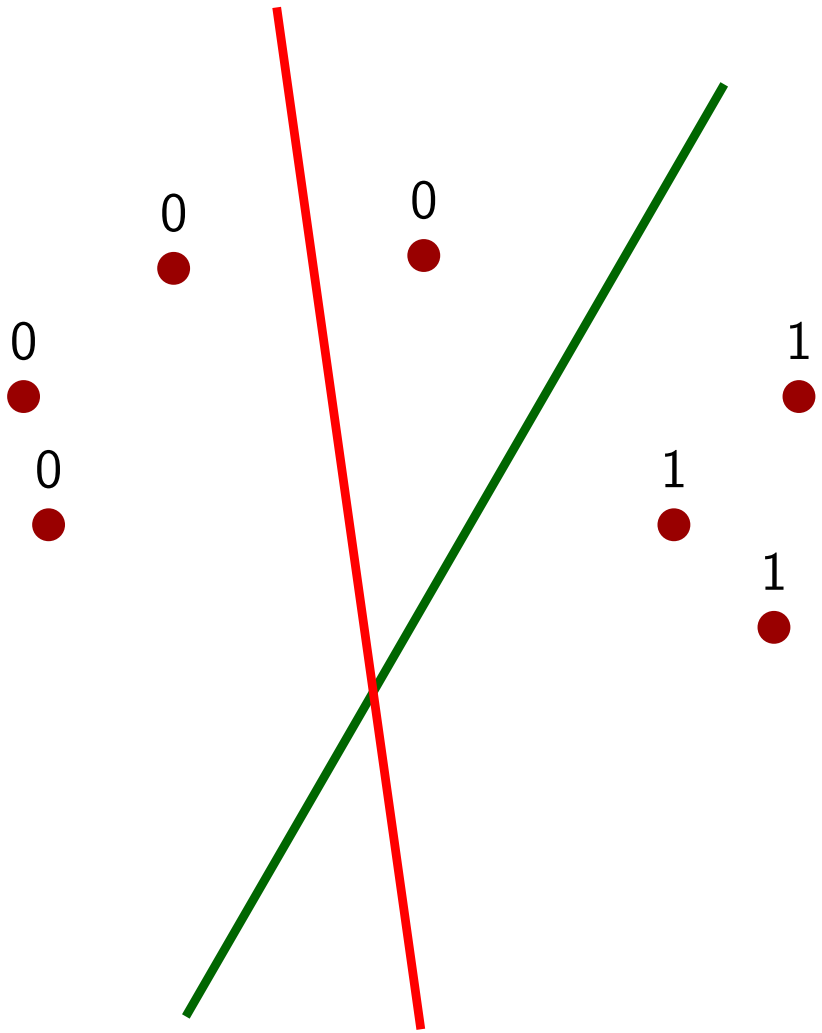
Given $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ we define an *augmented feature vector*

$$\tilde{\mathbf{x}} = (x_0, x_1, \ldots, x_n) \quad \text{where } x_0 = 1$$

This makes the notation for the linear classifier more succinct:

$$h[\vec{w}](\vec{x}) = sgn(\vec{w} \cdot \tilde{\mathbf{x}})$$

# Linear Classifier – Learning




- classification in the plane using a linear classifier
- if a point is incorrectly classified, the learning algorithm turns the line (hyperplane) to improve the classification

# Perceptron Learning

- Given a training set

$$D = \{(\vec{x}_1, c_1), (\vec{x}_2, c_2)), \ldots, (\vec{x}_p, c_p))\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $c_k \in \{0, 1\}$.

Recall that $\tilde{\mathbf{x}}_k = (x_{k0}, x_{k1} \ldots, x_{kn})$ where $x_{k0} = 1$.

- A weight vector $\vec{w} \in \mathbb{R}^{n+1}$ is **consistent with** $D$ if

$$h[\vec{w}](\vec{x}_k) = sgn(\vec{w} \cdot \tilde{\mathbf{x}}_k) = c_k \quad \text{for all } k = 1, \ldots, p$$

$D$ is **linearly separable** if there is a vector $\vec{w} \in \mathbb{R}^{n+1}$ which is consistent with $D$.

- Our goal is to find a consistent $\vec{w}$ assuming that $D$ is linearly separable.

# Perceptron – Learning Algorithm

**Online learning algorithm:**

Idea: Cyclically go through the training examples in $D$ and adapt weights. Whenever an example is incorrectly classified, turn the hyperplane so that the example becomes closer to its correct half-space.

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is randomly initialized close to $\vec{0} = (0, \ldots, 0)$
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

$$\begin{aligned} \vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \left( h[\vec{w}^{(t)}](\vec{x}_k) - c_k \right) \cdot \tilde{\mathbf{x}}_k \\ &= \vec{w}^{(t)} - \varepsilon \cdot \left( sgn\left( \vec{w}^{(t)} \cdot \tilde{\mathbf{x}}_k \right) - c_k \right) \cdot \tilde{\mathbf{x}}_k \end{aligned}$$

Here $k = (t \mod p) + 1$, i.e., the examples are considered cyclically, and $0 < \varepsilon \leq 1$ is a **learning rate**.

## Theorem (Rosenblatt)

*If $D$ is linearly separable, then there is $t^*$ such that $\vec{w}^{(t^*)}$ is consistent with $D$.*

# Example

Training set:

$$D = \{((2,-1),1),((2,1),1),((1,3),0)\}$$

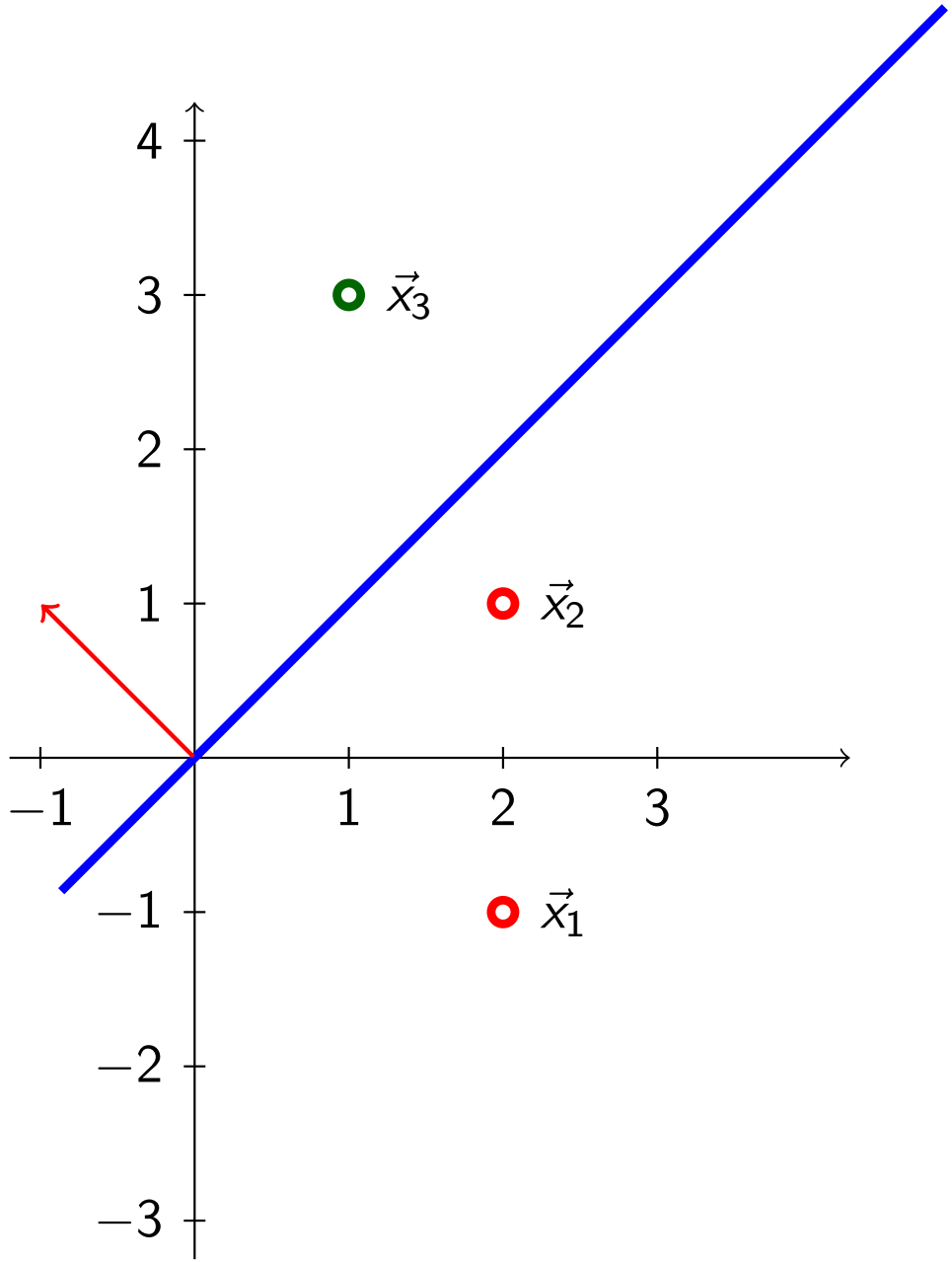
That is

$$\begin{aligned} \vec{x}_1 &= (2,-1) \\ \vec{x}_2 &= (2,1) \\ \vec{x}_3 &= (1,3) \end{aligned} \qquad \begin{aligned} \tilde{\mathbf{x}}_1 &= (1,2,-1) \\ \tilde{\mathbf{x}}_2 &= (1,2,1) \\ \tilde{\mathbf{x}}_3 &= (1,1,3) \end{aligned}$$

$$\begin{aligned} c_1 &= 1 \\ c_2 &= 1 \\ c_3 &= 0 \end{aligned}$$

Assume that the initial vector $\vec{w}^{(0)}$ is $\vec{w}^{(0)} = (0,-1,1)$.
Consider $\varepsilon = 1$.

# Example: Separating by $\vec{w}^{(0)}$




Denoting $\vec{w}^{(0)} = (w_0, w_1, w_2) = (0, -1, 1)$ the blue separating line is given by $w_0 + w_1 x_1 + w_2 x_2 = 0$.

The red vector normal to the blue line is $(w_1, w_2)$.

The points on the side of $(w_1, w_2)$ are assigned 1 by the classifier, the others zero. (In this case $\vec{x}_3$ is assigned one and $\vec{x}_1, \vec{x}_2$ are assigned zero, all of this is inconsistent with $c_1 = 1, c_2 = 1, c_3 = 0$.)

# Example: Computing $\vec{w}^{(1)}$

We have

$$\vec{w}^{(0)} \cdot \tilde{\mathbf{x}}_1 = (0, -1, 1) \cdot (1, 2, -1) = 0 - 2 - 1 = -3$$

thus

$$sgn\left(\vec{w}^{(0)} \cdot \tilde{\mathbf{x}}_1\right) = 0$$

and thus

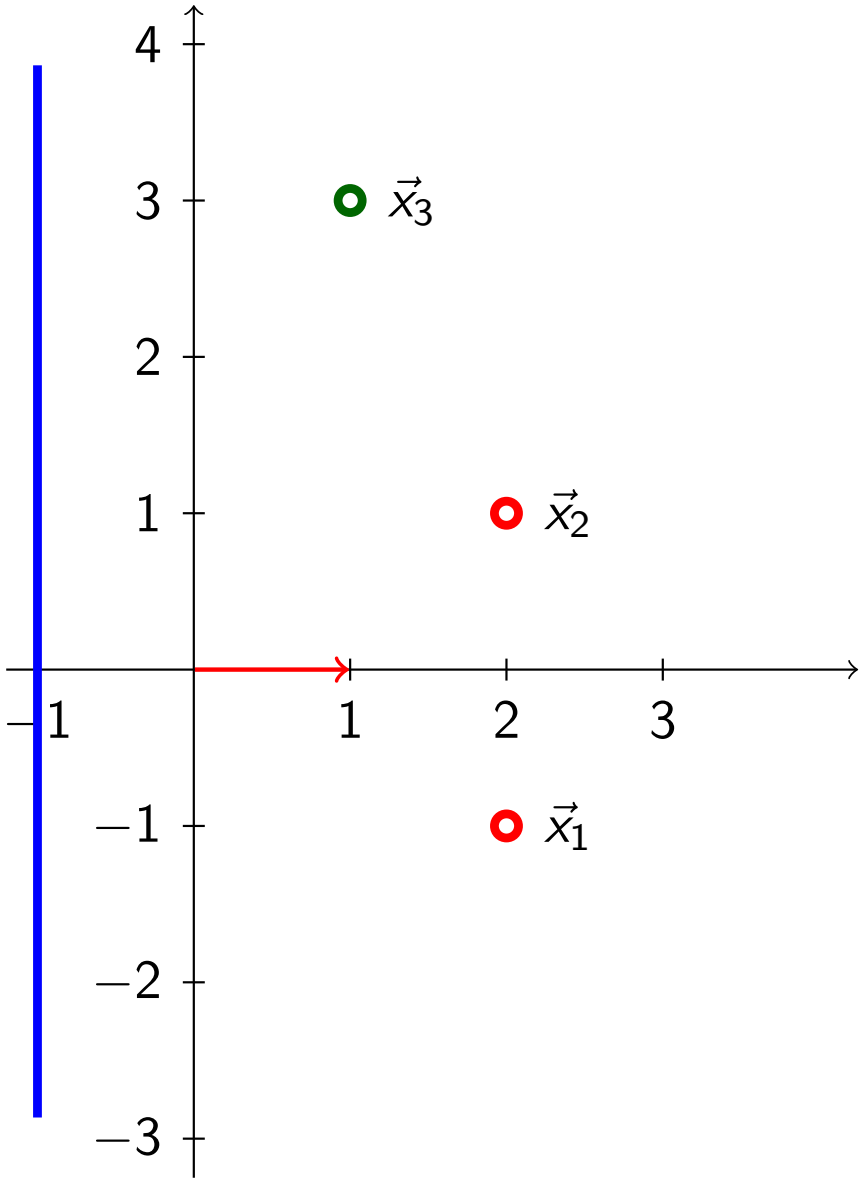
$$sgn\left(\vec{w}^{(0)} \cdot \tilde{\mathbf{x}}_1\right) - c_1 = 0 - 1 = -1$$

(I.e., $\vec{x}_1$ is not correctly classified, and $\vec{w}^{(0)}$ is not consistent with $D$.)

Hence,

$$\begin{aligned}
\vec{w}^{(1)} &= \vec{w}^{(0)} - \left(sgn\left(\vec{w}^{(0)} \cdot \tilde{\mathbf{x}}_1\right) - c_1\right) \cdot \tilde{\mathbf{x}}_1 \\
&= \vec{w}^{(0)} + \tilde{\mathbf{x}}_1 \\
&= (0, -1, 1) + (1, 2, -1) \\
&= (1, 1, 0)
\end{aligned}$$

# Example: Separating by $\vec{w}^{(1)}$



# Example: Computing $\vec{w}^{(2)}$

We have

$$\vec{w}^{(1)} \cdot \tilde{\mathbf{x}}_2 = (1, 1, 0) \cdot (1, 2, 1) = 1 + 2 = 3$$

thus

$$sgn\left(\vec{w}^{(1)} \cdot \tilde{\mathbf{x}}_2\right) = 1$$

and thus

$$sgn\left(\vec{w}^{(1)} \cdot \tilde{\mathbf{x}}_2\right) - c_2 = 1 - 1 = 0$$

(I.e., $\vec{x}_2$ *is* currently correctly classified by $\vec{w}^{(1)}$. However, as we will see, $\vec{x}_3$ is not well classified.)

Hence,

$$\vec{w}^{(2)} = \vec{w}^{(1)} = (1, 1, 0)$$

# Example: Computing $\vec{w}^{(3)}$

We have

$$\vec{w}^{(2)} \cdot \tilde{\mathbf{x}}_3 = (1,1,0) \cdot (1,1,3) = 1 + 1 = 2$$

thus

$$sgn\left(\vec{w}^{(2)} \cdot \tilde{\mathbf{x}}_3\right) = 1$$

and thus

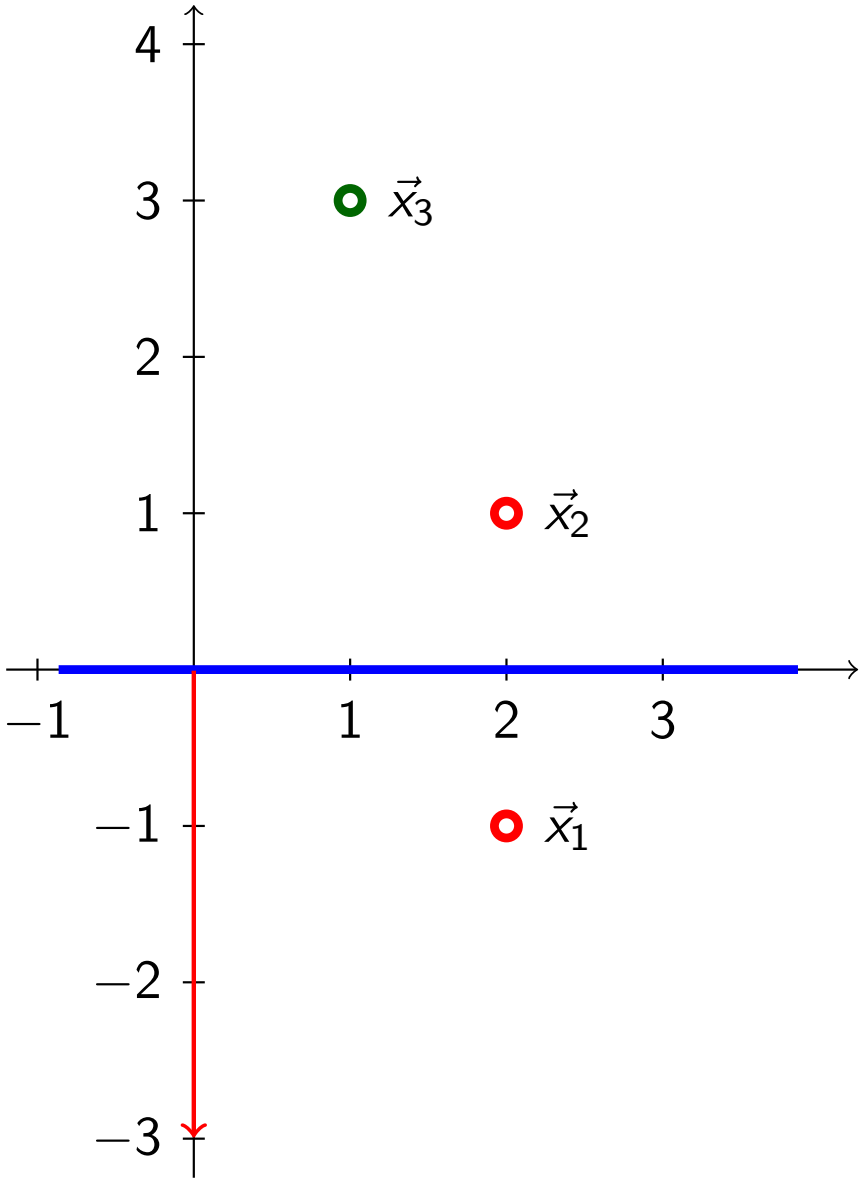
$$sgn\left(\vec{w}^{(2)} \cdot \tilde{\mathbf{x}}_3\right) - c_3 = 1 - 0 = 1$$

(This means that $\vec{x}_3$ is not well classified, and $\vec{w}^{(2)}$ is not consistent with $D$.)

Hence,

$$\begin{aligned}
\vec{w}^{(3)} &= \vec{w}^{(2)} - \left(sgn\left(\vec{w}^{(2)} \cdot \tilde{\mathbf{x}}_3\right) - c_3\right) \cdot \tilde{\mathbf{x}}_3 \\
&= \vec{w}^{(2)} - \tilde{\mathbf{x}}_3 \\
&= (1,1,0) - (1,1,3) \\
&= (0,0,-3)
\end{aligned}$$

# Example: Separating by $\vec{w}^{(3)}$



# Example: Computing $\vec{w}^{(4)}$

We have

$$\vec{w}^{(3)} \cdot \tilde{\mathbf{x}}_1 = (0, 0, -3) \cdot (1, 2, -1) = 3$$

thus

$$sgn\left(\vec{w}^{(3)} \cdot \tilde{\mathbf{x}}_1\right) = 1$$

and thus

$$sgn\left(\vec{w}^{(3)} \cdot \tilde{\mathbf{x}}_1\right) - c_1 = 1 - 1 = 0$$

(I.e., $\vec{x}_1$ *is* currently correctly classified by $\vec{w}^{(3)}$. However, we shall see that $\vec{x}_2$ is not.)

Hence,

$$\vec{w}^{(4)} = \vec{w}^{(3)} = (0, 0, -3)$$

# Example: Computing $\vec{w}^{(5)}$

We have

$$\vec{w}^{(4)} \cdot \tilde{\mathbf{x}}_2 = (0, 0, -3) \cdot (1, 2, 1) = -3$$

thus

$$sgn\left(\vec{w}^{(4)} \cdot \tilde{\mathbf{x}}_2\right) = 0$$

and thus

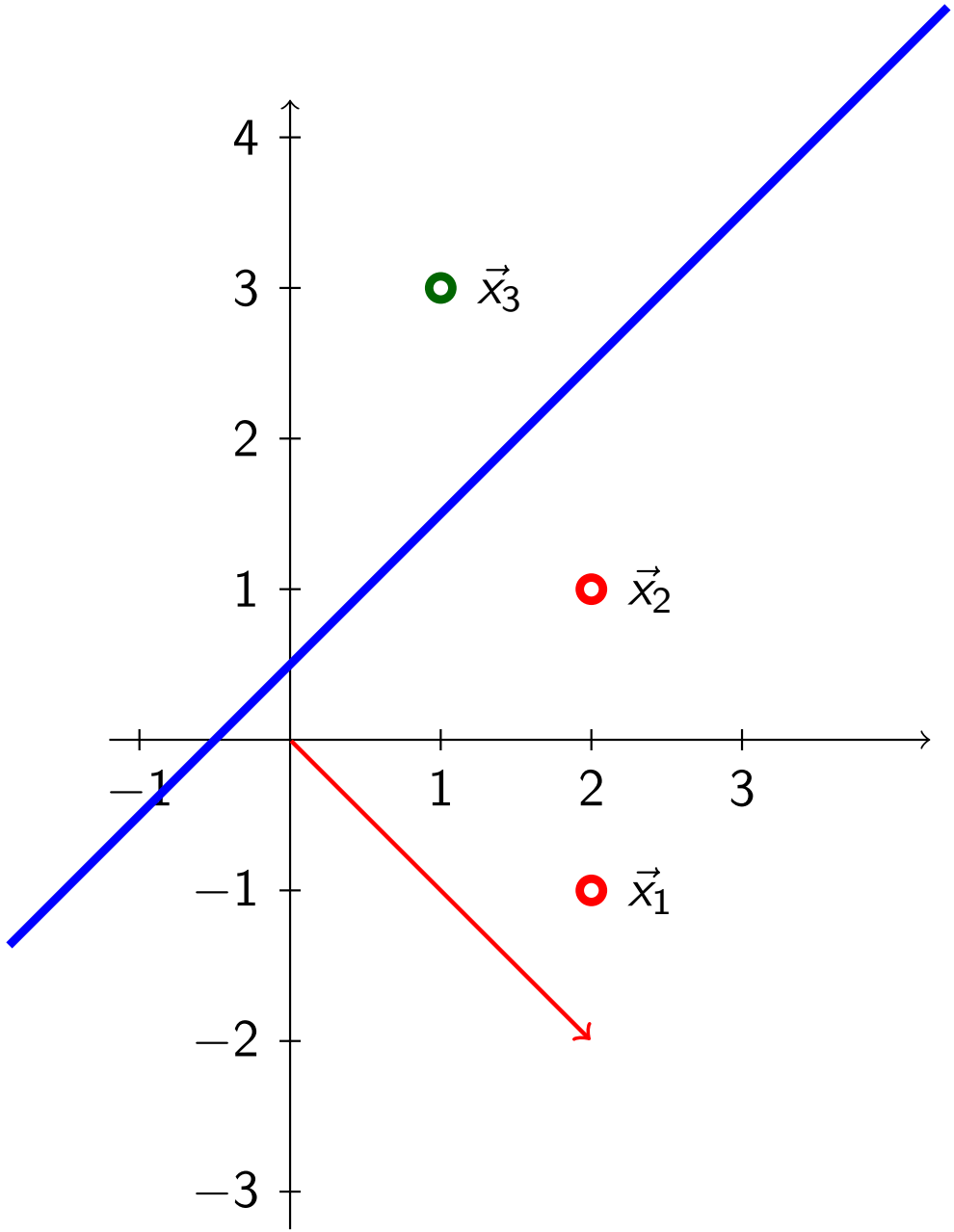
$$sgn\left(\vec{w}^{(4)} \cdot \tilde{\mathbf{x}}_2\right) - c_2 = 0 - 1 = -1$$

(I.e., $\vec{x}_2$ is not correctly classified, and $\vec{w}^{(4)}$ is not consistent with $D$.)

Hence,

$$\begin{aligned}
\vec{w}^{(5)} &= \vec{w}^{(4)} - \left(sgn\left(\vec{w}^{(4)} \cdot \tilde{\mathbf{x}}_2\right) - c_2\right) \cdot \tilde{\mathbf{x}}_2 \\
&= \vec{w}^{(4)} + \tilde{\mathbf{x}}_2 \\
&= (0, 0, -3) + (1, 2, 1) \\
&= (1, 2, -2)
\end{aligned}$$

# Example: Separating by $\vec{w}^{(5)}$



# Example: The result

The vector $\vec{w}^{(5)}$ is consistent with $D$:

$$sgn\left(\vec{w}^{(5)} \cdot \tilde{\mathbf{x}}_1\right) = sgn\left((1,2,-2) \cdot (1,2,-1)\right) = sgn(7) = 1 = c_1$$

$$sgn\left(\vec{w}^{(5)} \cdot \tilde{\mathbf{x}}_2\right) = sgn\left((1,2,-2) \cdot (1,2,1)\right) = sgn(3) = 1 = c_2$$

$$sgn\left(\vec{w}^{(5)} \cdot \tilde{\mathbf{x}}_3\right) = sgn\left((1,2,-2) \cdot (1,1,3)\right) = sgn(-3) = 0 = c_3$$

# Perceptron – Learning Algorithm

**Batch learning algorithm:**

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is randomly initialized close to $\vec{0} = (0, \ldots, 0)$
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

$$\begin{aligned} \vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left( h[\vec{w}^{(t)}](\vec{x}_k) - c_k \right) \cdot \tilde{\mathbf{x}}_k \\ &= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left( sgn\left( \vec{w}^{(t)} \cdot \tilde{\mathbf{x}}_k \right) - c_k \right) \cdot \tilde{\mathbf{x}}_k \end{aligned}$$

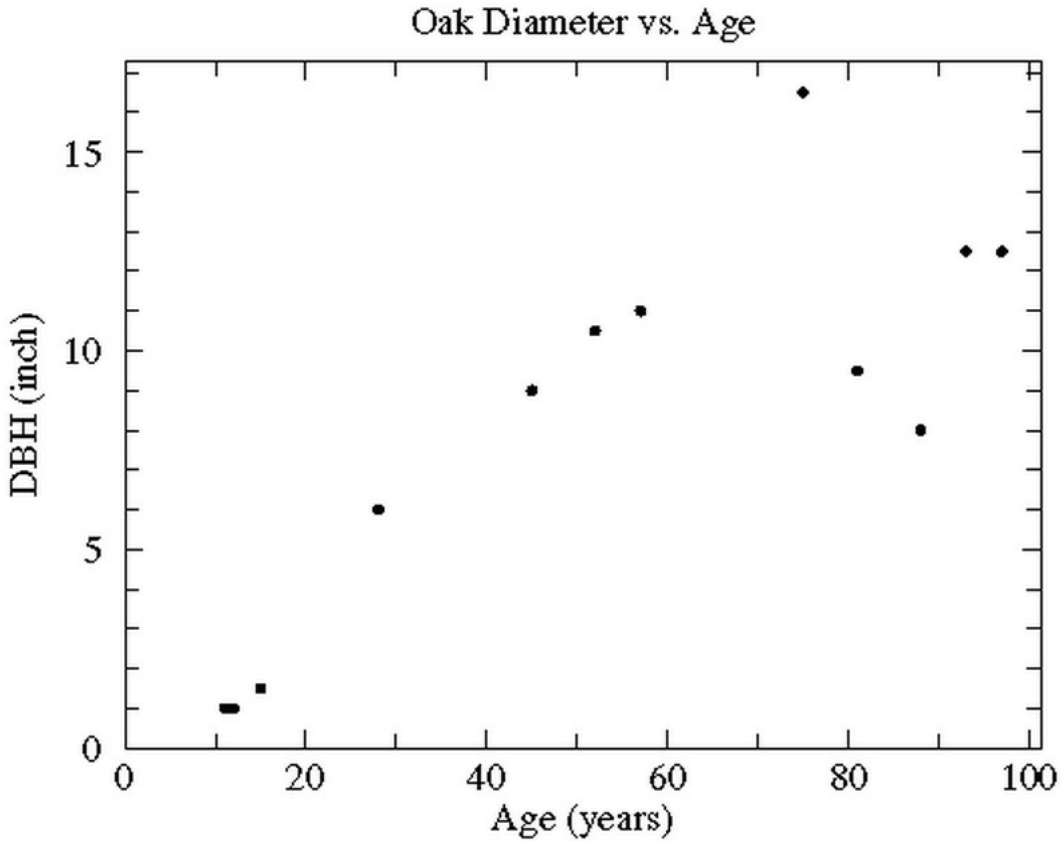
Here $0 < \varepsilon \leq 1$ is a **learning rate**.

# Linear Regression

# Linear Regression – Oaks in Wisconsin

This example is from *How to Lie with Statistics* by Darrell Huff (1954)

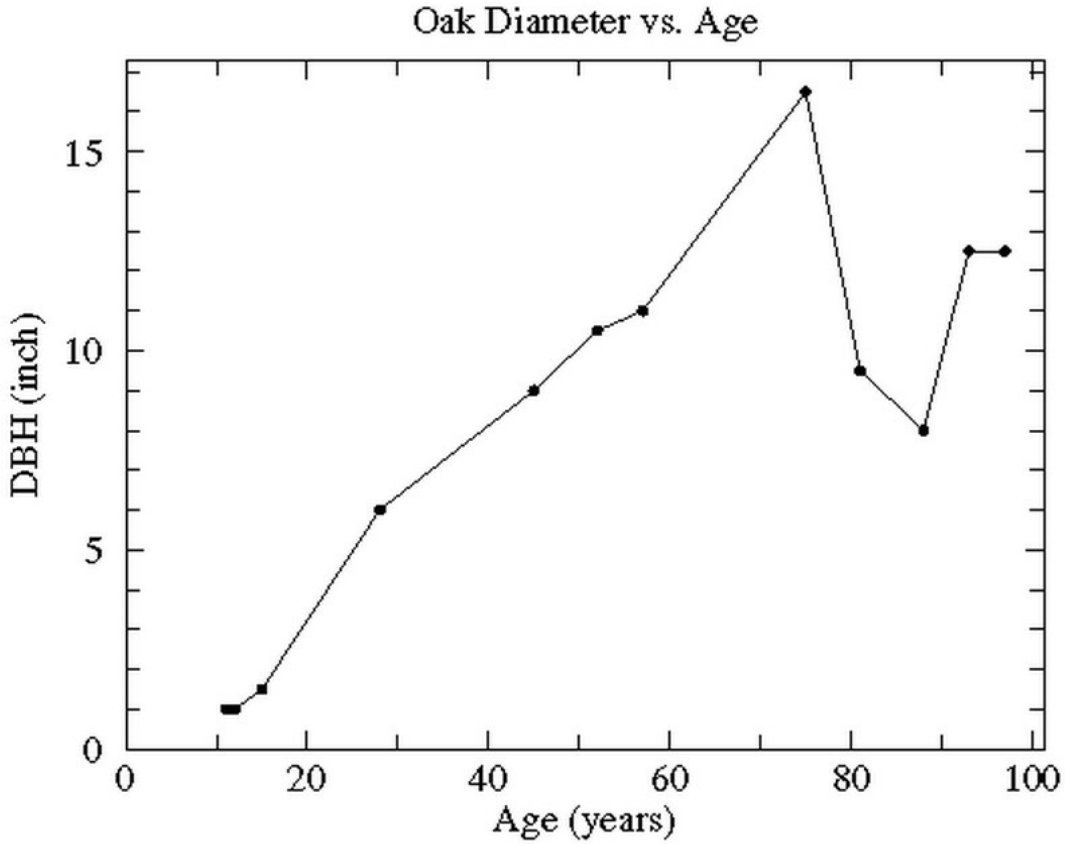
| **Age (years)** | ***DBH* (inch)** |
|---|---|
| 97 | 12.5 |
| 93 | 12.5 |
| 88 | 8.0 |
| 81 | 9.5 |
| 75 | 16.5 |
| 57 | 11.0 |
| 52 | 10.5 |
| 45 | 9.0 |
| 28 | 6.0 |
| 15 | 1.5 |
| 12 | 1.0 |
| 11 | 1.0 |



# Linear Regression – Oaks in Wisconsin

This example is from *How to Lie with Statistics* by Darrell Huff (1954)

| Age (years) | *DBH* (inch) |
|---|---|
| 97 | 12.5 |
| 93 | 12.5 |
| 88 | 8.0 |
| 81 | 9.5 |
| 75 | 16.5 |
| 57 | 11.0 |
| 52 | 10.5 |
| 45 | 9.0 |
| 28 | 6.0 |
| 15 | 1.5 |
| 12 | 1.0 |
| 11 | 1.0 |

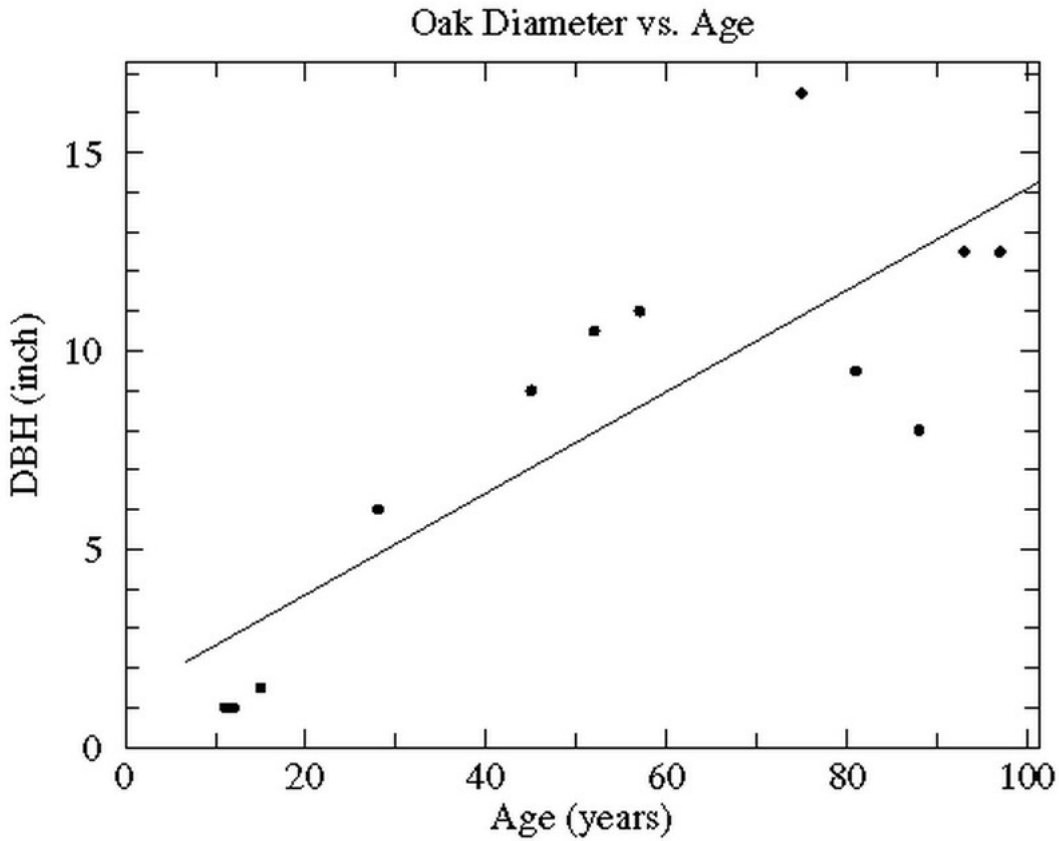



NO!

# Linear Regression – Oaks in Wisconsin

This example is from *How to Lie with Statistics* by Darrell Huff (1954)

| Age (years) | *DBH* (inch) |
|---|---|
| 97 | 12.5 |
| 93 | 12.5 |
| 88 | 8.0 |
| 81 | 9.5 |
| 75 | 16.5 |
| 57 | 11.0 |
| 52 | 10.5 |
| 45 | 9.0 |
| 28 | 6.0 |
| 15 | 1.5 |
| 12 | 1.0 |
| 11 | 1.0 |




possibly YES!

# Linear Regression

**Our goal:**

- Given a set $D$ of training examples of the form $(\vec{x}, f)$ where $\vec{x} \in \mathbb{R}^n$ and $f \in \mathbb{R}$,
- construct a model function $h : \mathbb{R}^n \to \mathbb{R}$ such that

$$h(\vec{x}) \approx f \text{ for all training examples } (\vec{x}, f) \in D$$

Here $\approx$ means that the values are somewhat close to each other w.r.t. an appropriate *error function* $E$.

In what follows we use the *squared error* defined by

$$E = \frac{1}{2} \sum_{(\vec{x},f)\in D} (h(\vec{x}) - f)^2$$

Our goal is to minimize $E$.

The main reason is that this function has nice mathematical properties (as opposed, e.g., to $\sum_{(\vec{x},f)\in D} |h(\vec{x}) - f|$ ).

# Linear Function Approximation

- Given a set $D$ of training examples:

$$D = \{(\vec{x}_1, f_1), (\vec{x}_2, f_2), \ldots, (\vec{x}_p, f_p)\}$$

  Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $f_k \in \mathbb{R}$.

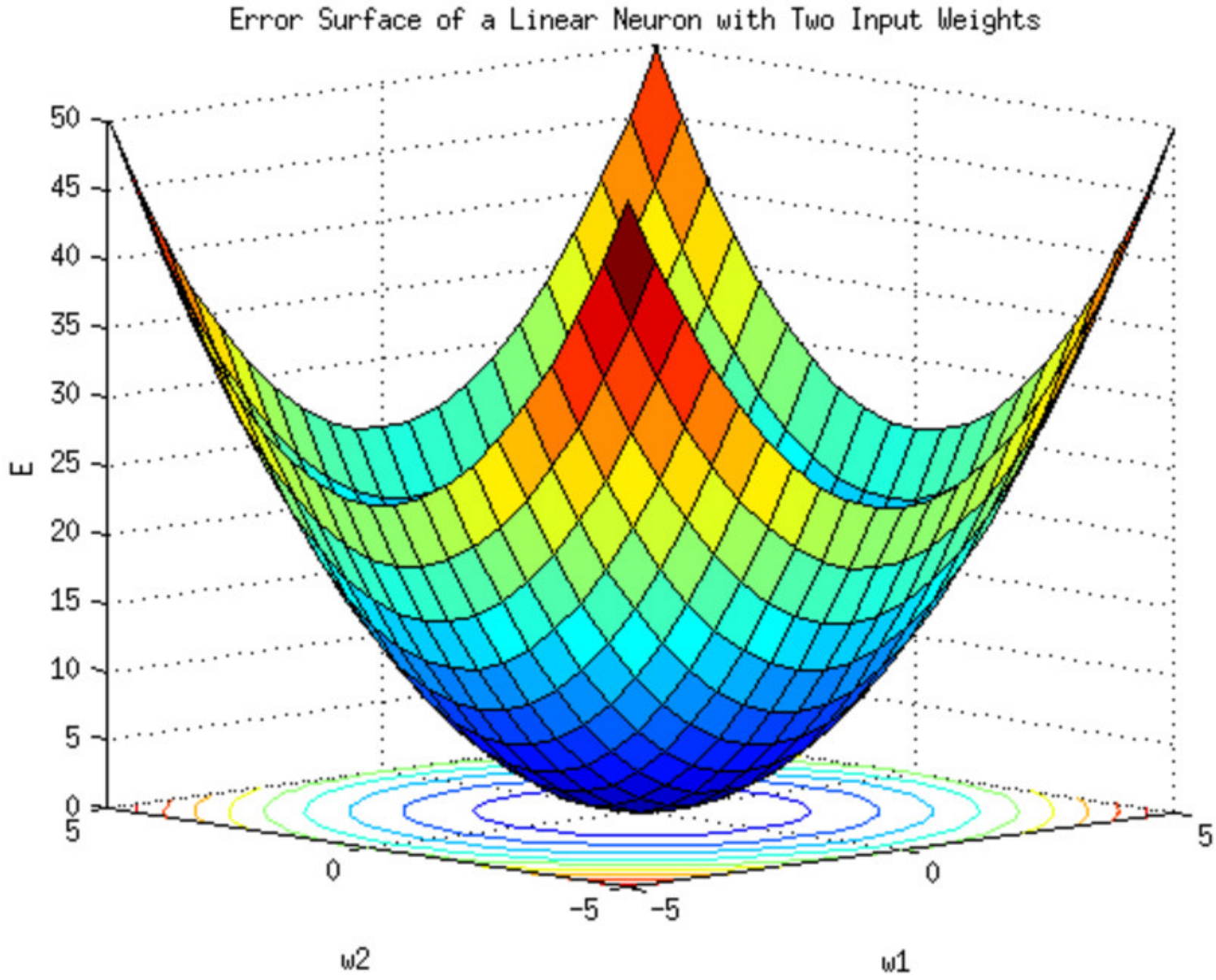
- **Our goal:** Find $\vec{w}$ so that $h[\vec{w}](\vec{x_k}) = \vec{w} \cdot \tilde{\mathbf{x}}_k$ is close to $f_k$ for every $k = 1, \ldots, p$.

  Recall that $\tilde{\mathbf{x}}_k = (x_{k0}, x_{k1} \ldots, x_{kn})$ where $x_{k0} = 1$.

- **Squared Error Function:**

$$E(\vec{w}) = \frac{1}{2}\sum_{k=1}^{p}(\vec{w} \cdot \tilde{\mathbf{x}}_k - f_k)^2 = \frac{1}{2}\sum_{k=1}^{p}\left(\sum_{i=0}^{n} w_i x_{ki} - f_k\right)^2$$

# Error function



# Gradient of the Error Function

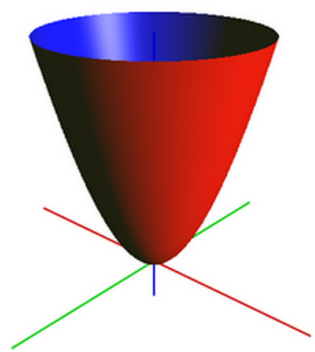
Consider the **gradient** of the error function:

$$\nabla E(\vec{w}) = \left( \frac{\partial E}{\partial w_0}(\vec{w}), \ldots, \frac{\partial E}{\partial w_n}(\vec{w}) \right) = \sum_{k=1}^{p} (\vec{w} \cdot \tilde{\mathbf{x}}_k - f_k) \cdot \tilde{\mathbf{x}}_k$$

What is the gradient $\nabla E(\vec{w})$? It is a vector in $\mathbb{R}^{n+1}$ which points in the direction of the steepest *ascent* of $E$ (its length corresponds to the steepness). Note that here the vectors $\tilde{\mathbf{x}}_k$ are *fixed* parameters of $E$!

**Fact:**
If $\nabla E(\vec{w}) = \vec{0} = (0, \ldots, 0)$, then $\vec{w}$ is a global minimum of $E$.

This follows from the fact that $E$ is a convex paraboloid that has a unique extreme, which is a minimum.



# Gradient of the error function

Consider $n = 1$, which means that $\vec{w} = (w_0, w_1)$ and we write $x$ instead of $\vec{x}$ since $\vec{x} \in \mathbb{R}^n = \mathbb{R}^1 = \mathbb{R}$.

Then the model is $h[\vec{w}](x) = w_0 + w_1 \cdot x$.

Consider a concrete training set:

$$\begin{aligned} \mathcal{T} &= \{(2,1),(3,2),(4,5)\} \\ &= \{(x_1,f_1),(x_2,f_2),(x_3,f_3)\} \end{aligned}$$

The augmented feature vectors are: $(1,2),(1,3),(1,4)$.

$$E(w_0,w_1) = \tfrac{1}{2}[(w_0+w_1\cdot 2-1)^2+(w_0+w_1\cdot 3-2)^2+(w_0+w_1\cdot 4-5)^2]$$

$$\frac{\partial E}{\partial w_0} = (w_0+w_1\cdot 2-1)\cdot 1+(w_0+w_1\cdot 3-2)\cdot 1+(w_0+w_1\cdot 4-5)\cdot 1$$
$$\frac{\partial E}{\partial w_1} = (w_0+w_1\cdot 2-1)\cdot 2+(w_0+w_1\cdot 3-2)\cdot 3+(w_0+w_1\cdot 4-5)\cdot 4$$

$$\nabla E(\vec{w}) = \left(\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}\right) =$$
$$(w_0+w_1\cdot 2-1)\cdot(1,2)+(w_0+w_1\cdot 3-2)\cdot(1,3)+(w_0+w_1\cdot 4-5)\cdot(1,4)$$

# Function Approximation – Learning

**Gradient Descent:**

- Weights $\vec{w}^{(0)}$ are initialized randomly close to $\vec{0}$.
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

$$\begin{aligned}
\vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \nabla E(\vec{w}^{(t)}) \\
&= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left( \vec{w}^{(t)} \cdot \tilde{\mathbf{x}}_k - f_k \right) \cdot \tilde{\mathbf{x}}_k \\
&= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left( h[\vec{w}^{(t)}](\vec{x}_k) - f_k \right) \cdot \tilde{\mathbf{x}}_k
\end{aligned}$$

Here $0 < \varepsilon \leq 1$ is a learning rate.

Note that the algorithm is almost similar to the batch perceptron algorithm!

## Proposition

*For sufficiently small $\varepsilon > 0$ the sequence $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$ converges (component-wisely) to the global minimum of $E$.*

Training set:

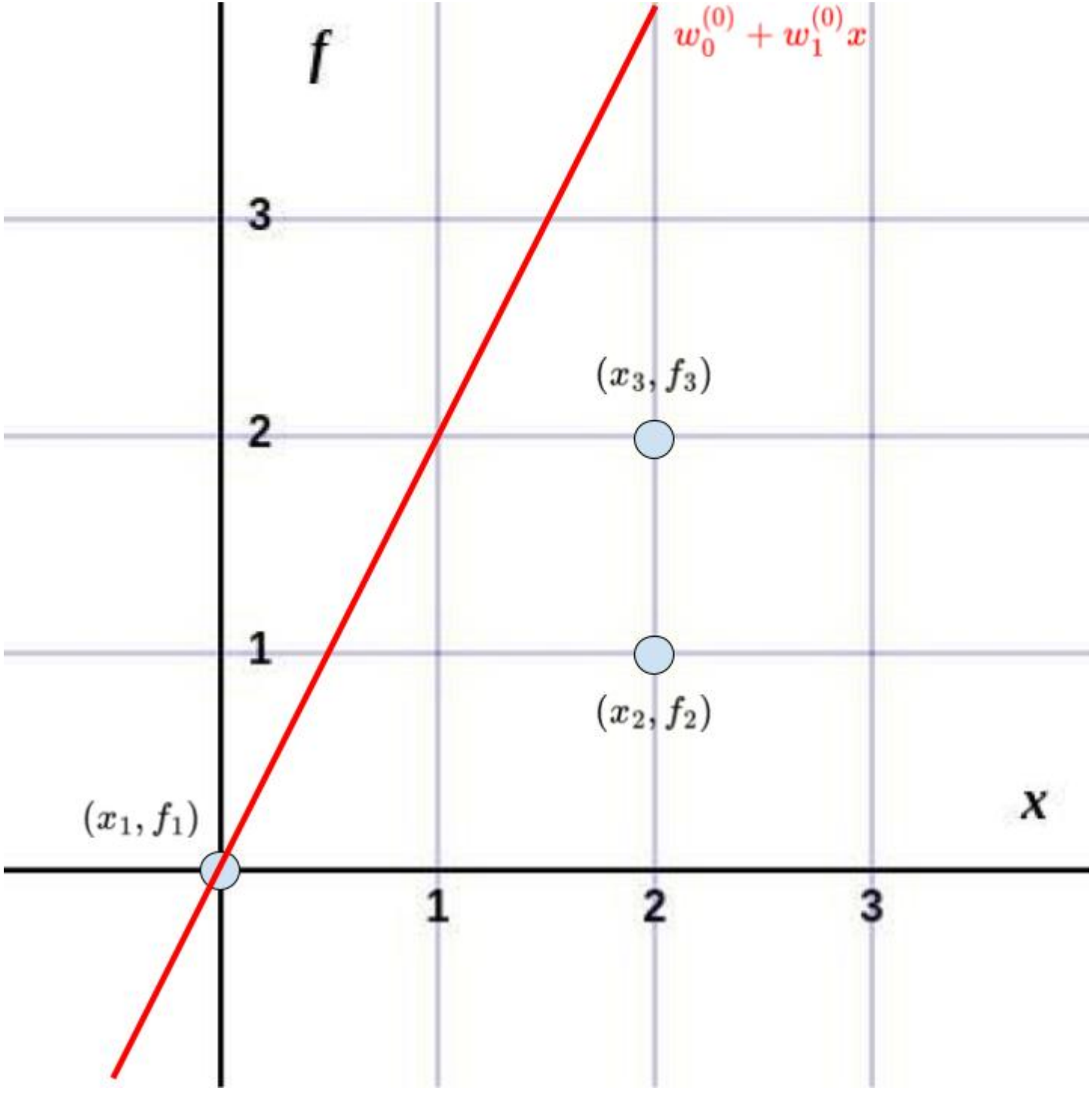
$$D = \{(x_1, f_1), (x_2, f_2), (x_3, f_3)\} = \{(0, 0), (2, 1), (2, 2)\}$$

Note that input vectors are one dimensional, so we write them as numbers.
That is

$$\begin{aligned} x_1 &= 0 \\ x_2 &= 2 \\ x_3 &= 2 \end{aligned} \qquad \begin{aligned} \tilde{\mathbf{x}}_1 &= (1, 0) \\ \tilde{\mathbf{x}}_2 &= (1, 2) \\ \tilde{\mathbf{x}}_3 &= (1, 2) \end{aligned}$$

$$\begin{aligned} f_1 &= 0 \\ f_2 &= 1 \\ f_3 &= 2 \end{aligned}$$

Assume that the initial vector $\vec{w}^{(0)}$ is $\vec{w}^{(0)} = (w_0^{(0)}, w_1^{(0)}) = (0, 2)$.
Consider $\varepsilon = \frac{1}{10}$.


f
$w_0^{(0)} + w_1^{(0)} x$
3
2
1
$(x_3, f_3)$
$(x_2, f_2)$
$(x_1, f_1)$
x
1
2
3

Training set:
$D = \{(x_1, f_1), (x_2, f_2), (x_3, f_3)\} = \{(0,0), (2,1), (2,2)\}$ Augmented input vectors: $\tilde{\mathbf{x}}_1 = (1,0),\ \tilde{\mathbf{x}}_2 = (1,2),\ \tilde{\mathbf{x}}_1 = (1,2)$

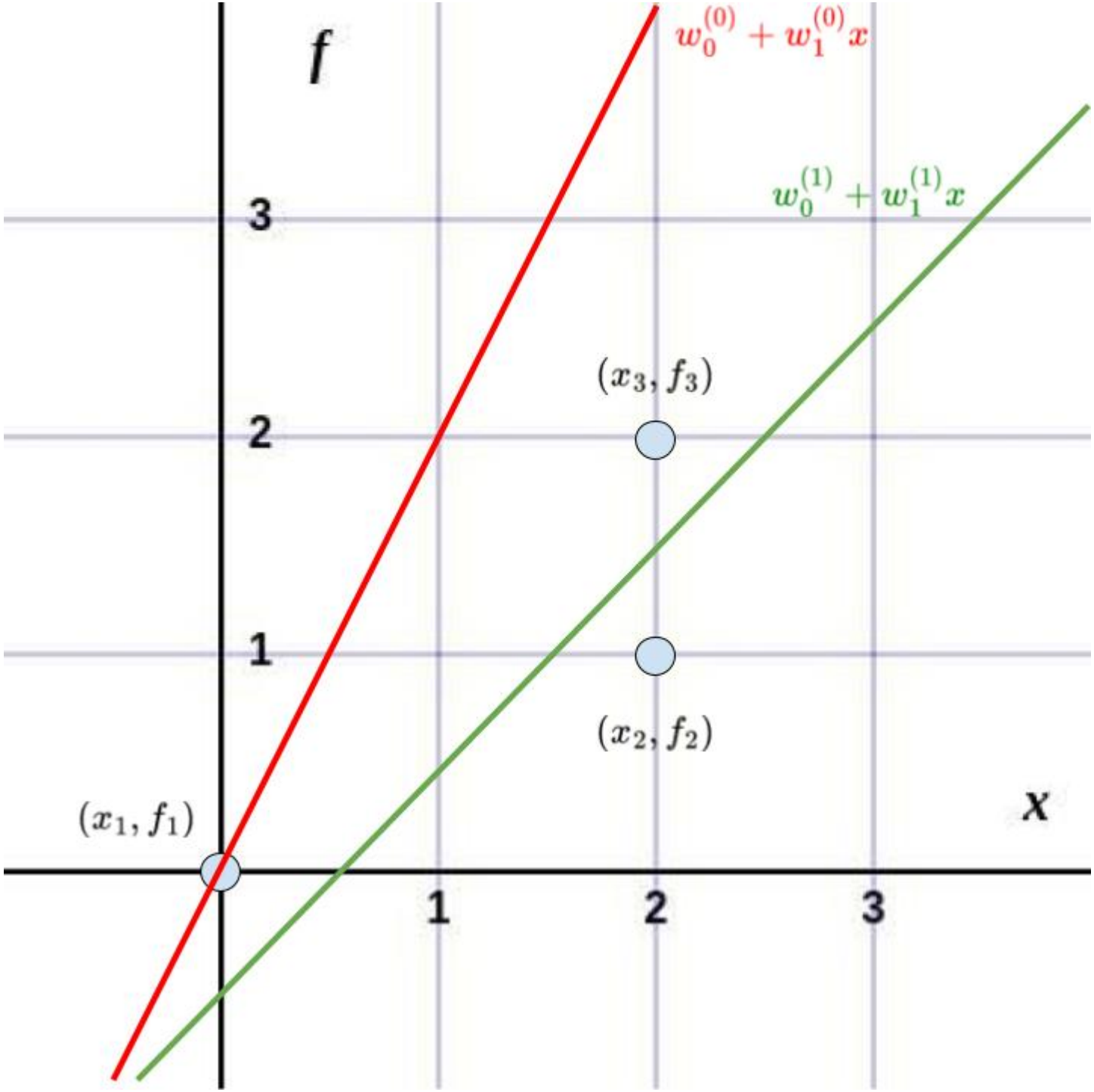
$$\nabla E(\vec{w}) = \left(\frac{\partial E}{\partial w_0}(\vec{w}), \frac{\partial E}{\partial w_1}(\vec{w})\right) = (w_0 + w_1 \cdot x_1 - f_1) \cdot \tilde{\mathbf{x}}_1 + (w_0 + w_1 \cdot x_2 - f_2) \cdot \tilde{\mathbf{x}}_2 + (w_0 + w_1 \cdot x_3 - f_3) \cdot \tilde{\mathbf{x}}_3$$

For $\vec{w}^{(0)} = (0,2)$ we have

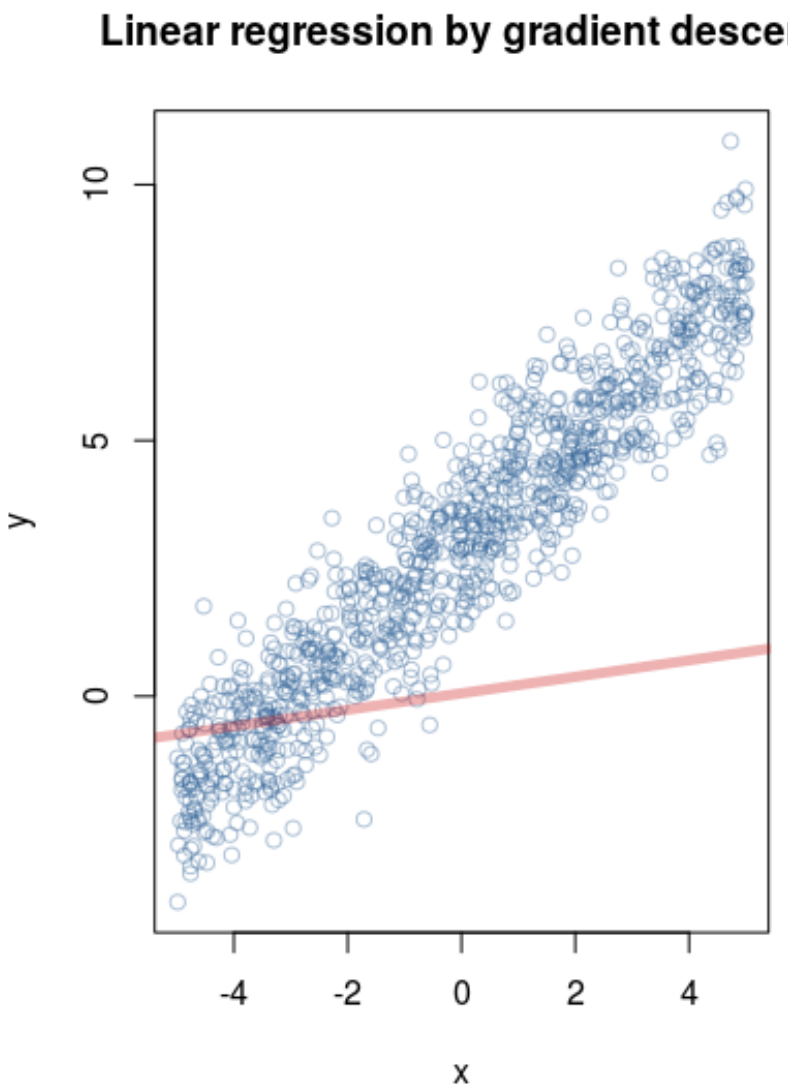
$$\begin{aligned}\nabla E(\vec{w}^{(0)}) =& (0 + 2 \cdot 0 - 0) \cdot (1,0) \\ &+ (0 + 2 \cdot 2 - 1) \cdot (1,2) \\ &+ (0 + 2 \cdot 2 - 2) \cdot (1,2) = (3,6) + (2,4) = (5,10)\end{aligned}$$

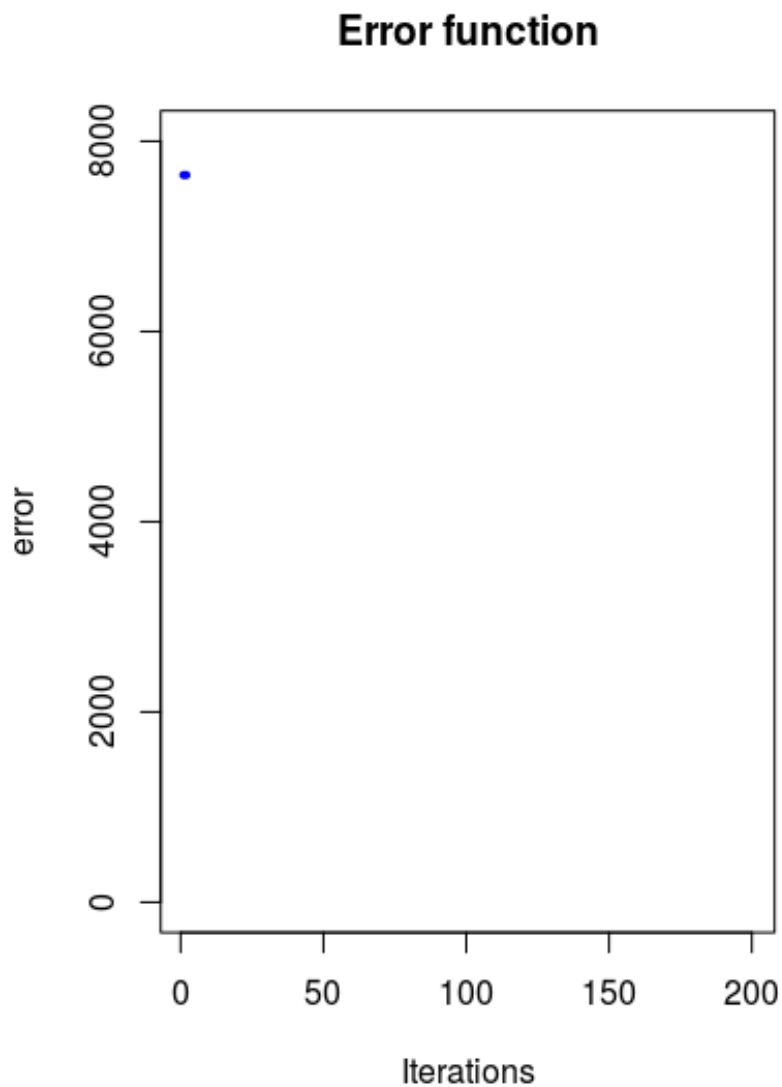
Finally, $\vec{w}^{(1)}$ is computed by

$$\vec{w}^{(1)} = \vec{w}^{(0)} - \varepsilon \cdot \nabla E(\vec{w}^{(0)}) = (0,2) - \frac{1}{10} \cdot (5,10) = (-1/2, 1)$$

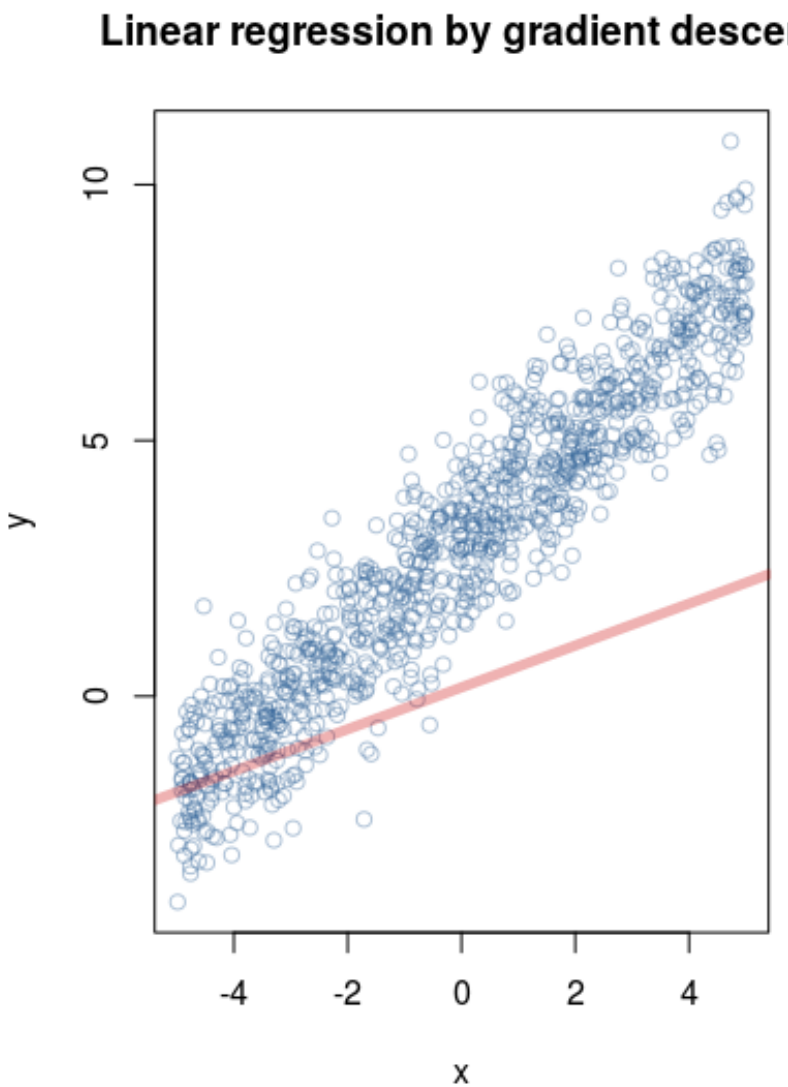

f
$w_0^{(0)} + w_1^{(0)} x$
$w_0^{(1)} + w_1^{(1)} x$
3
2
1
$(x_3, f_3)$
$(x_2, f_2)$
$(x_1, f_1)$
x
1
2
3

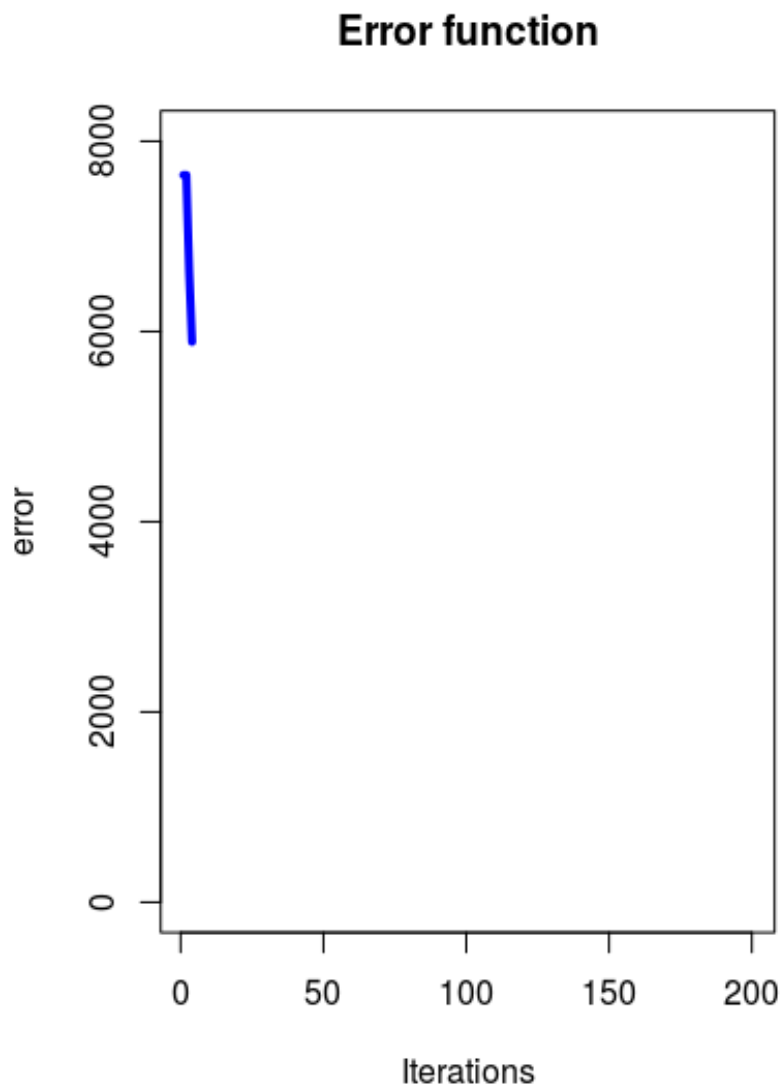
# Linear Regression - Animation



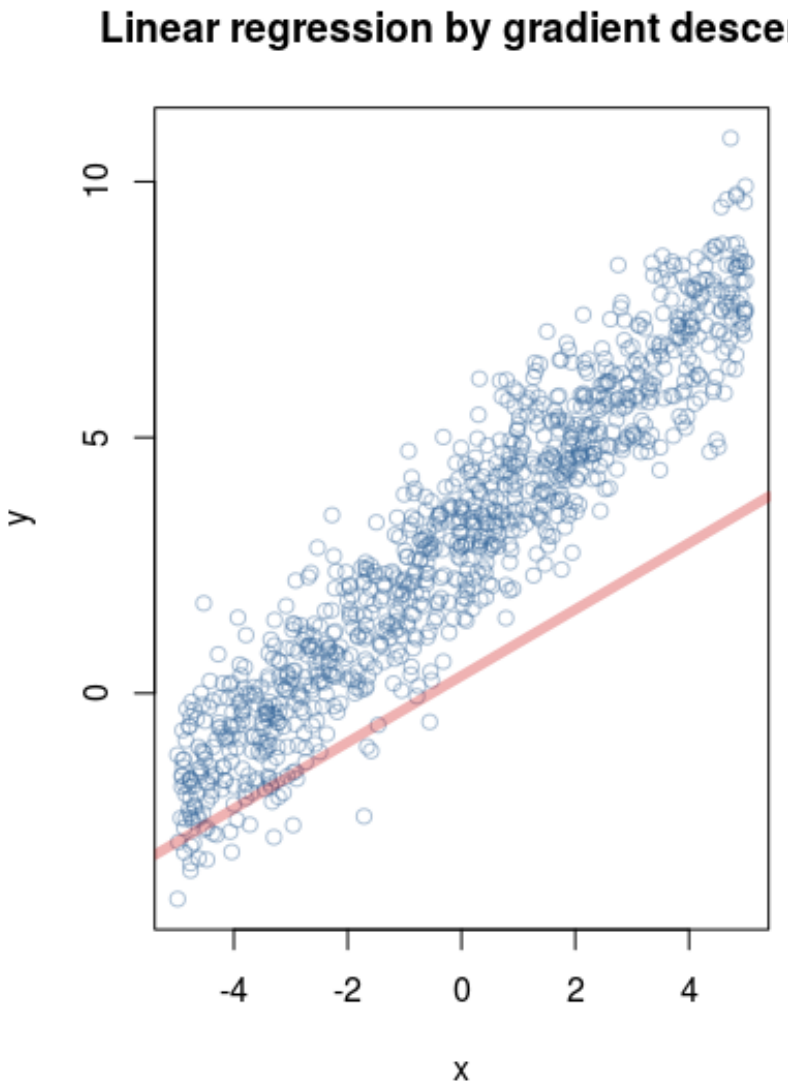



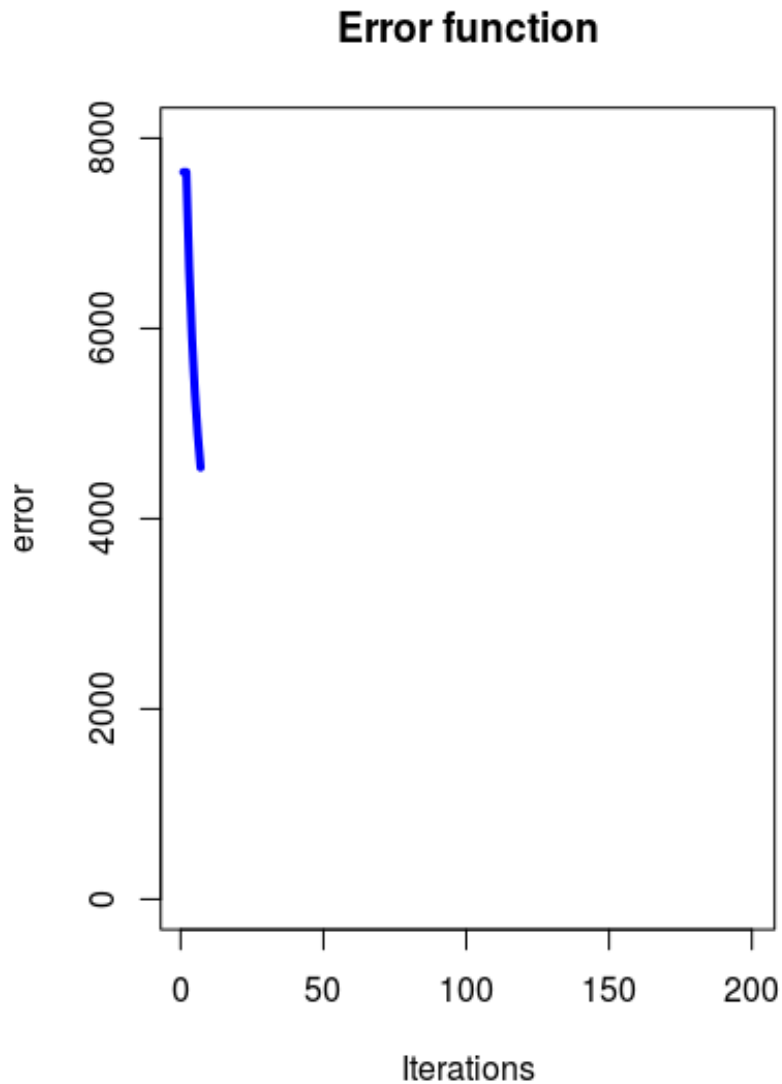
# Linear Regression - Animation



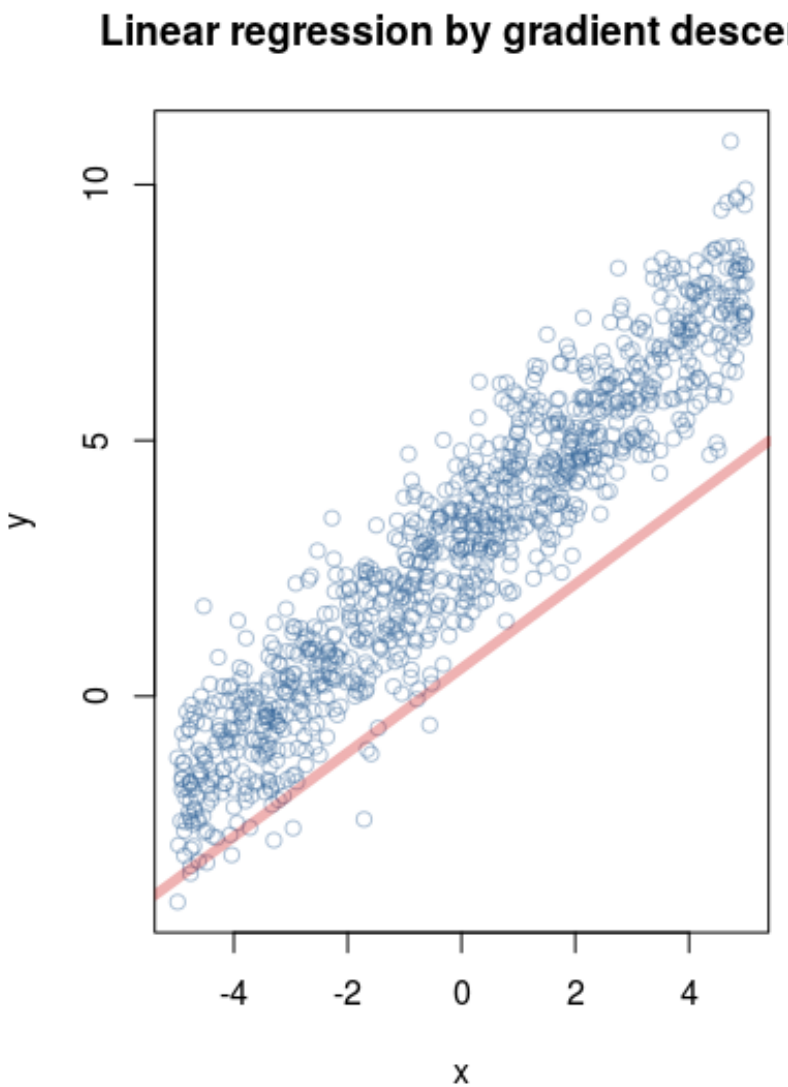



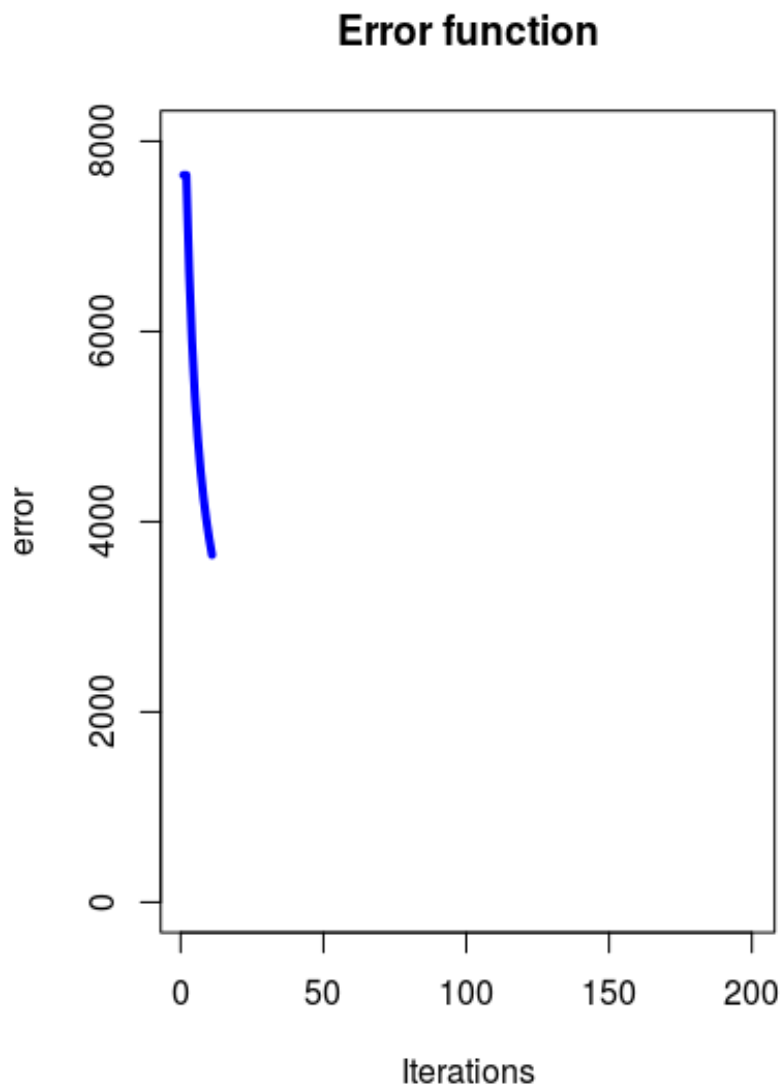
# Linear Regression - Animation



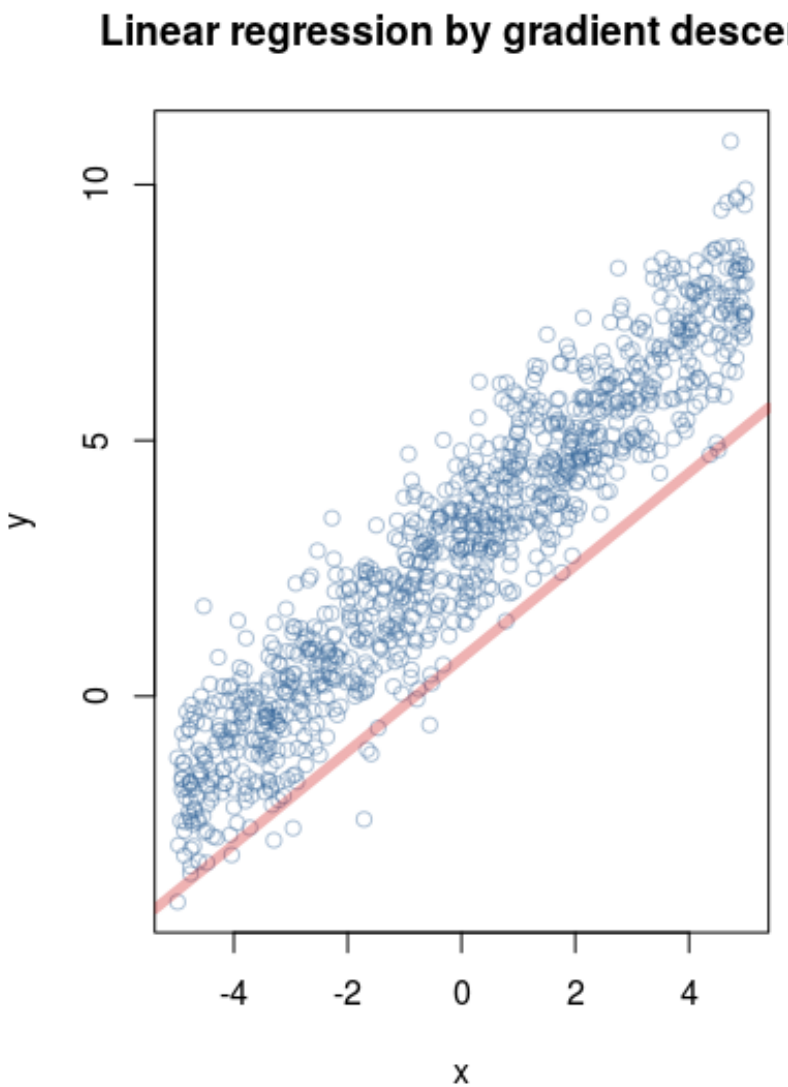



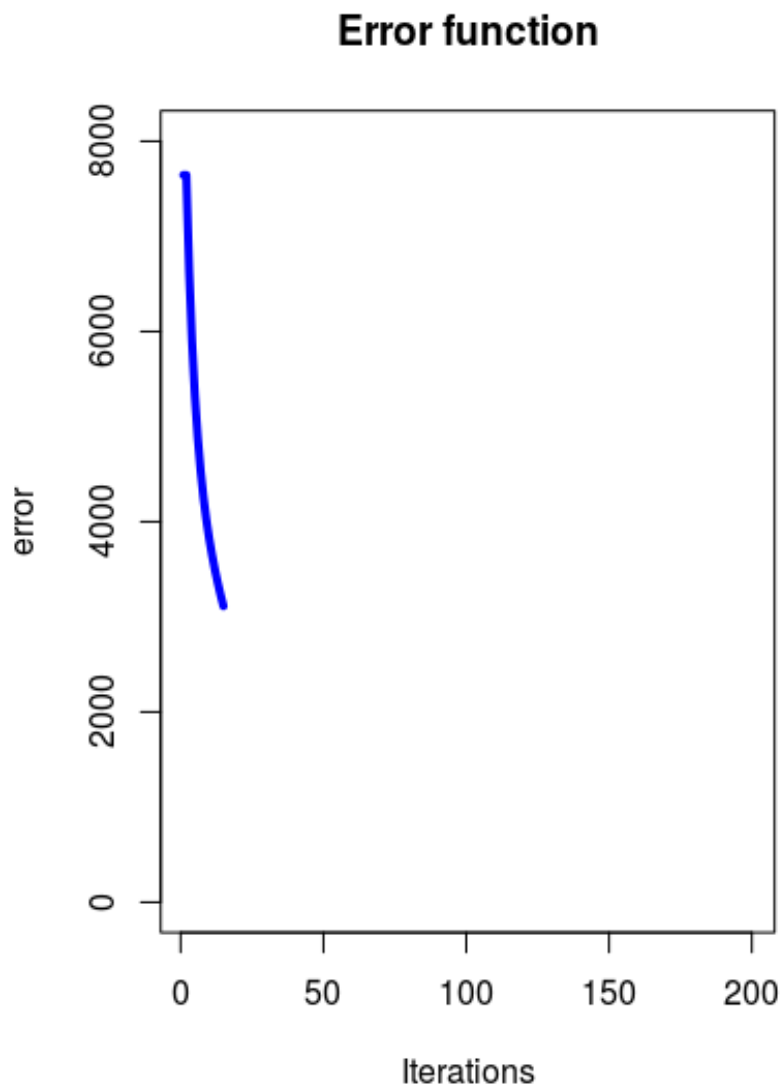
# Linear Regression - Animation



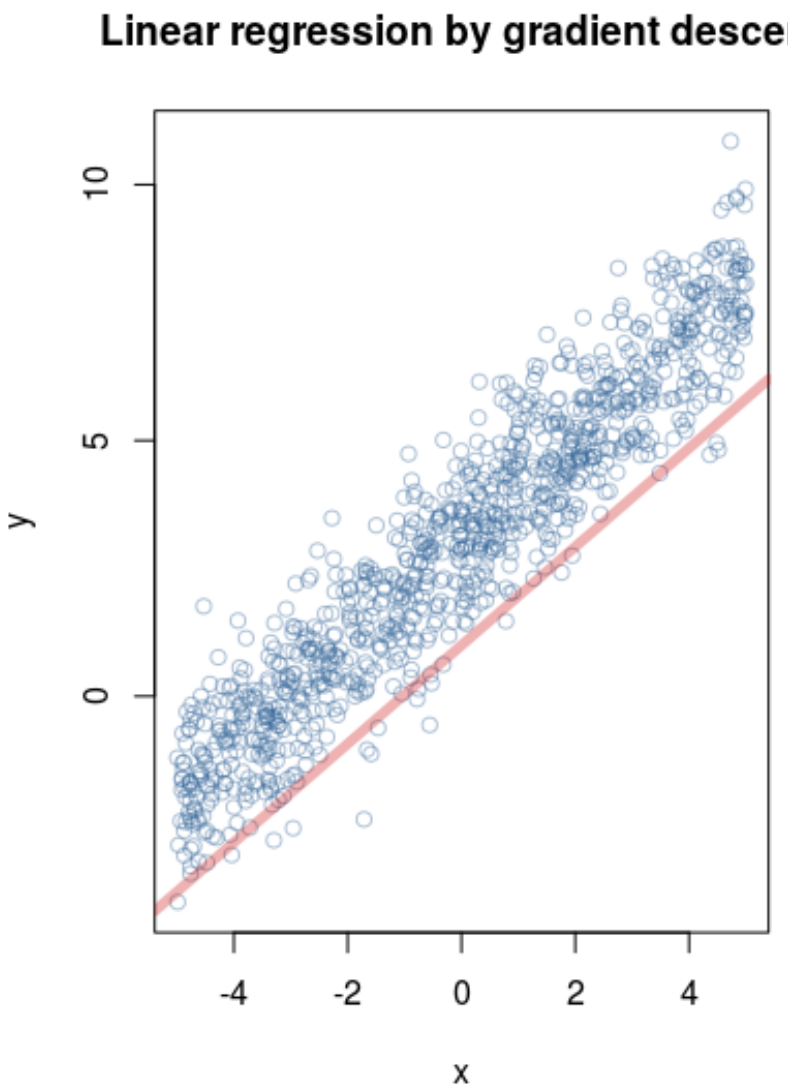



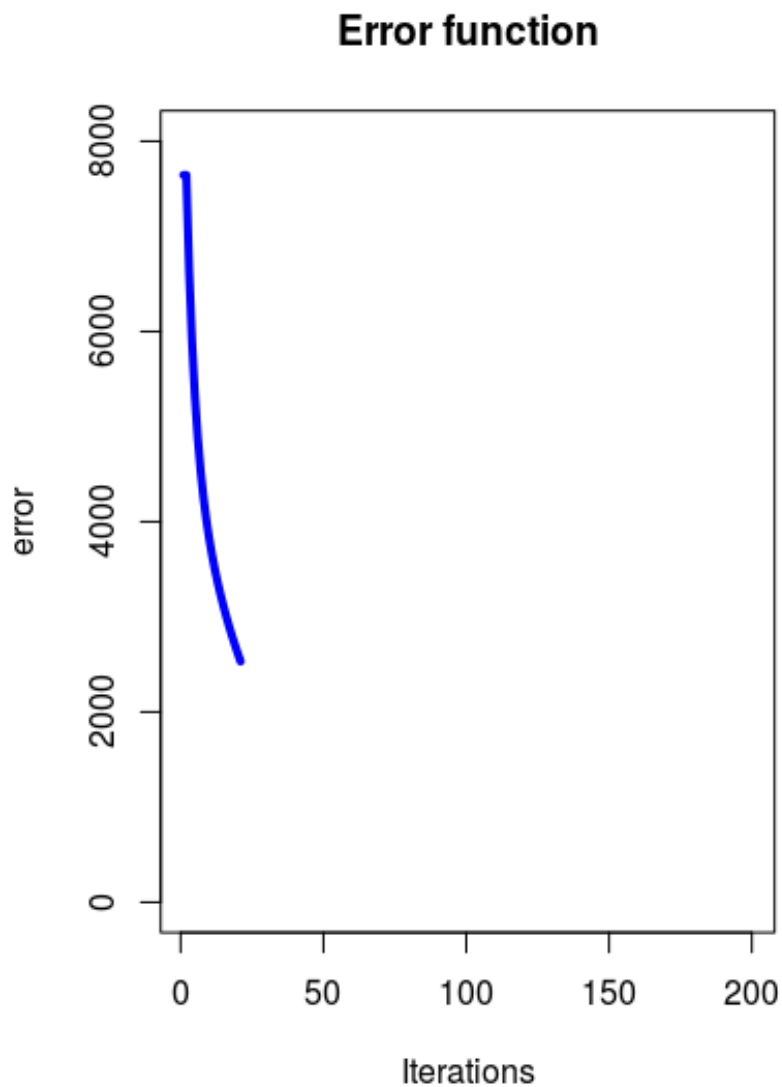
# Linear Regression - Animation



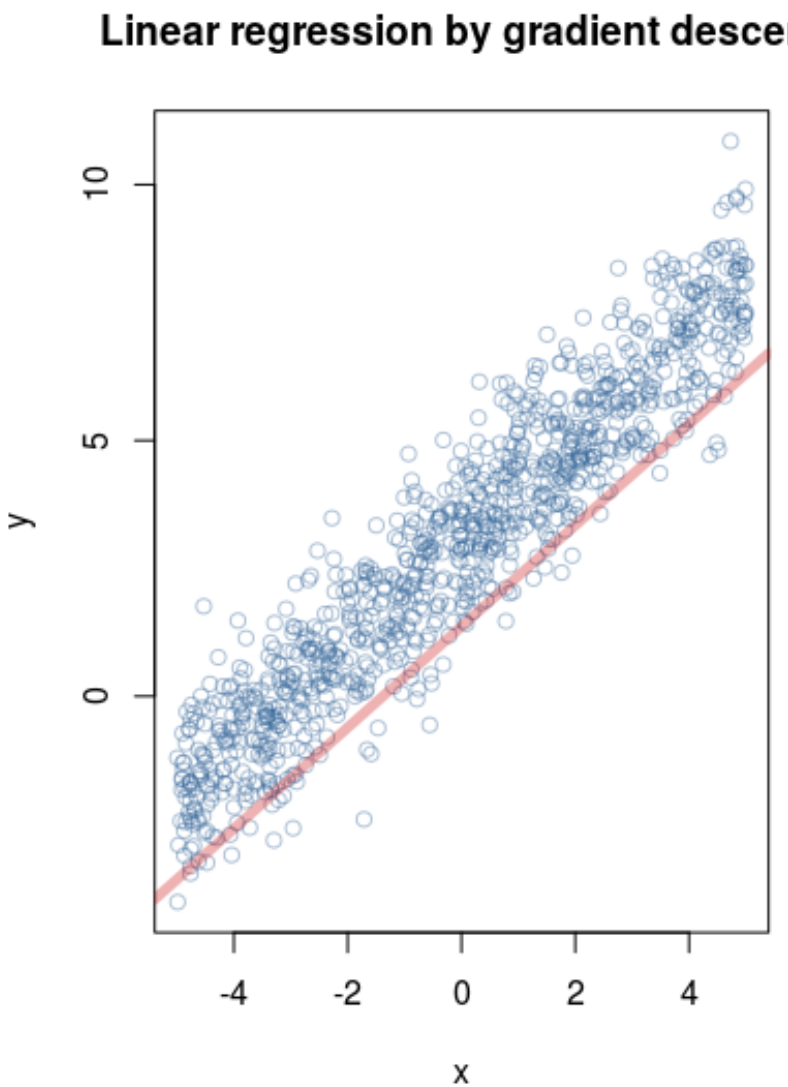



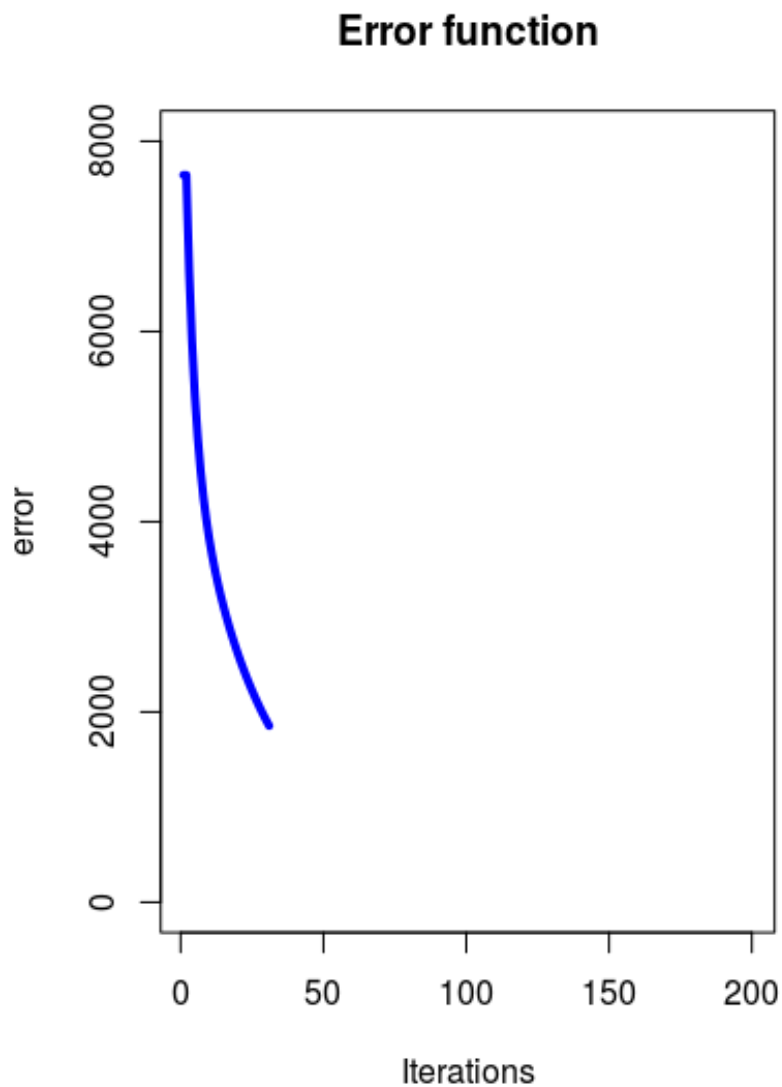
# Linear Regression - Animation



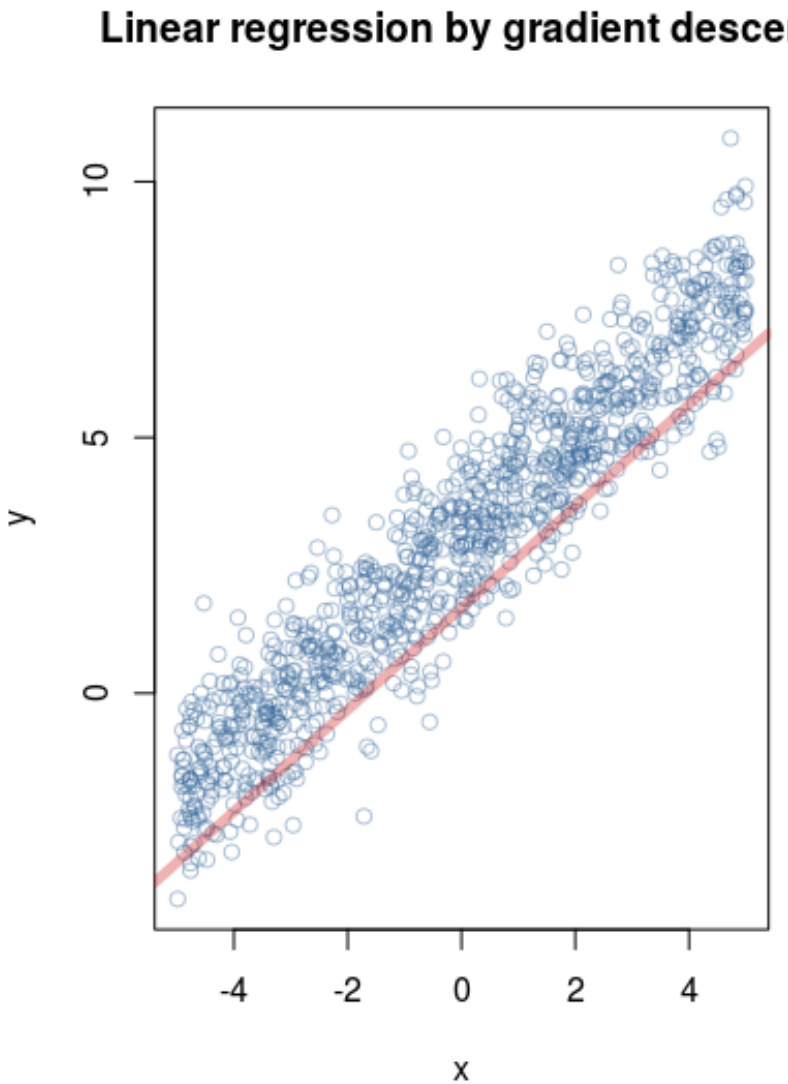



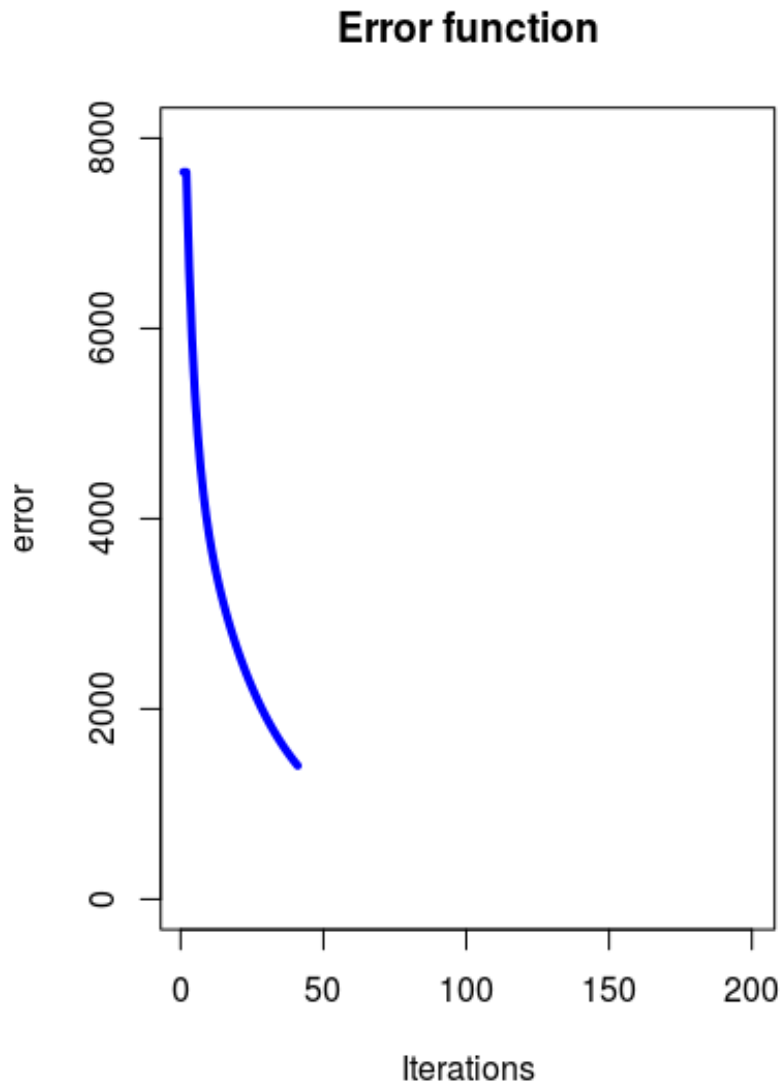
# Linear Regression - Animation



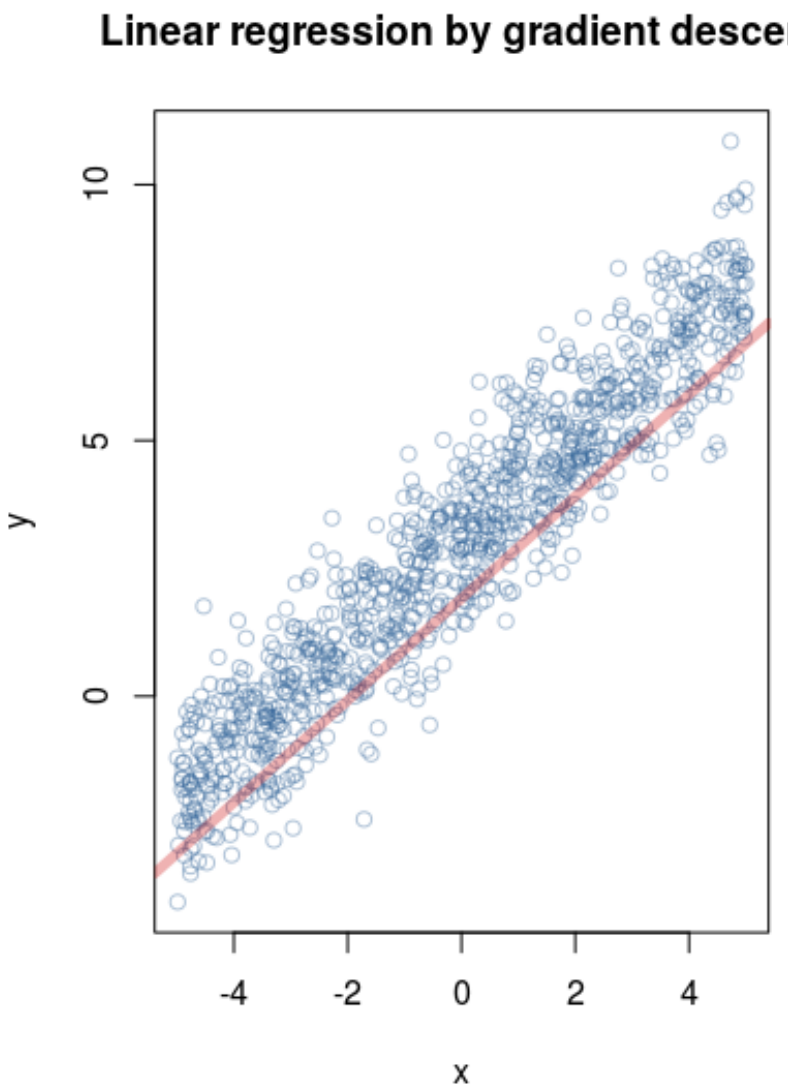



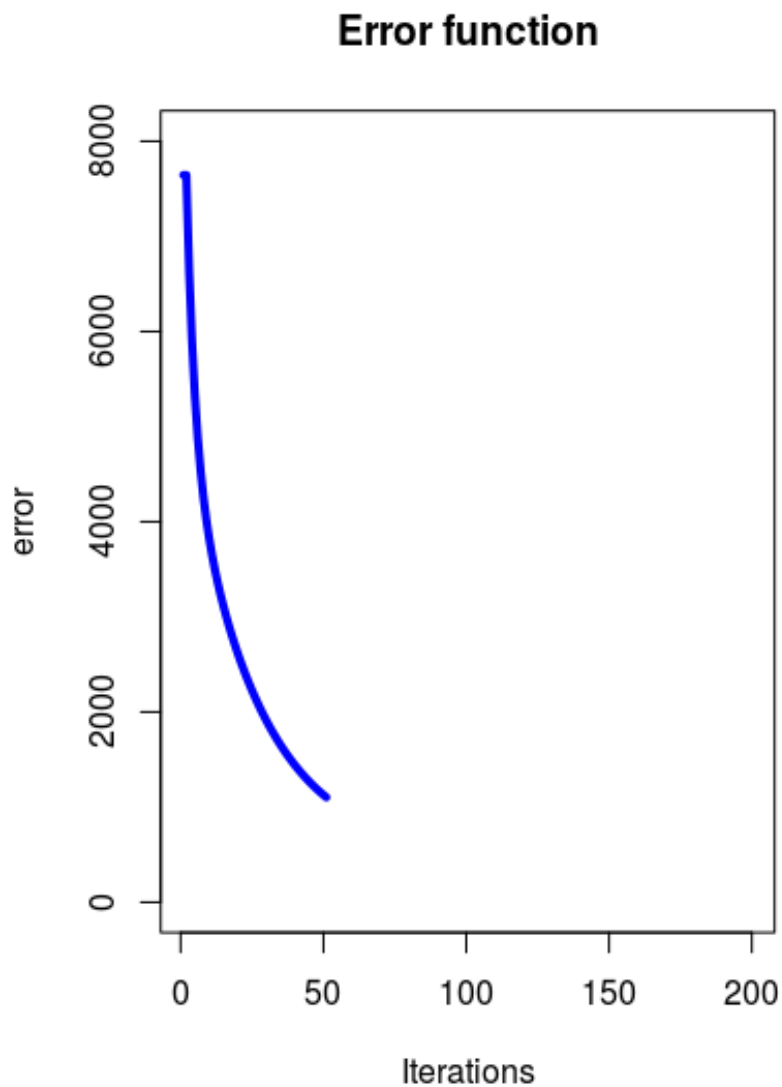
# Linear Regression - Animation



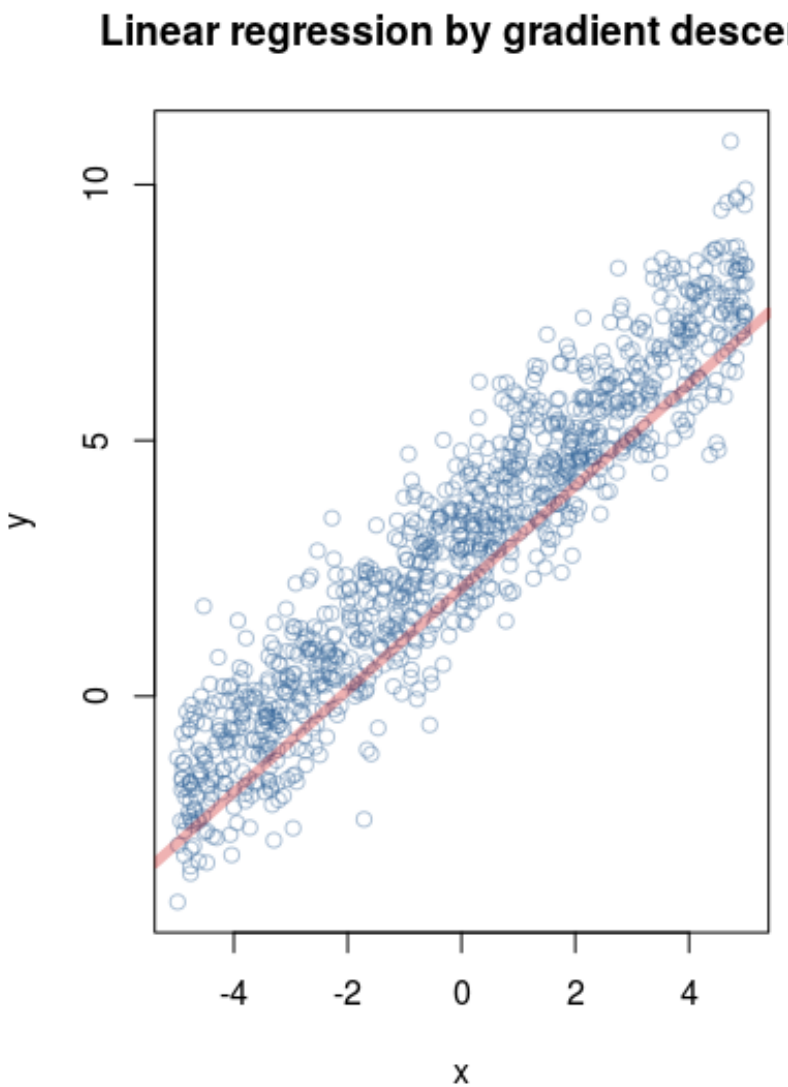



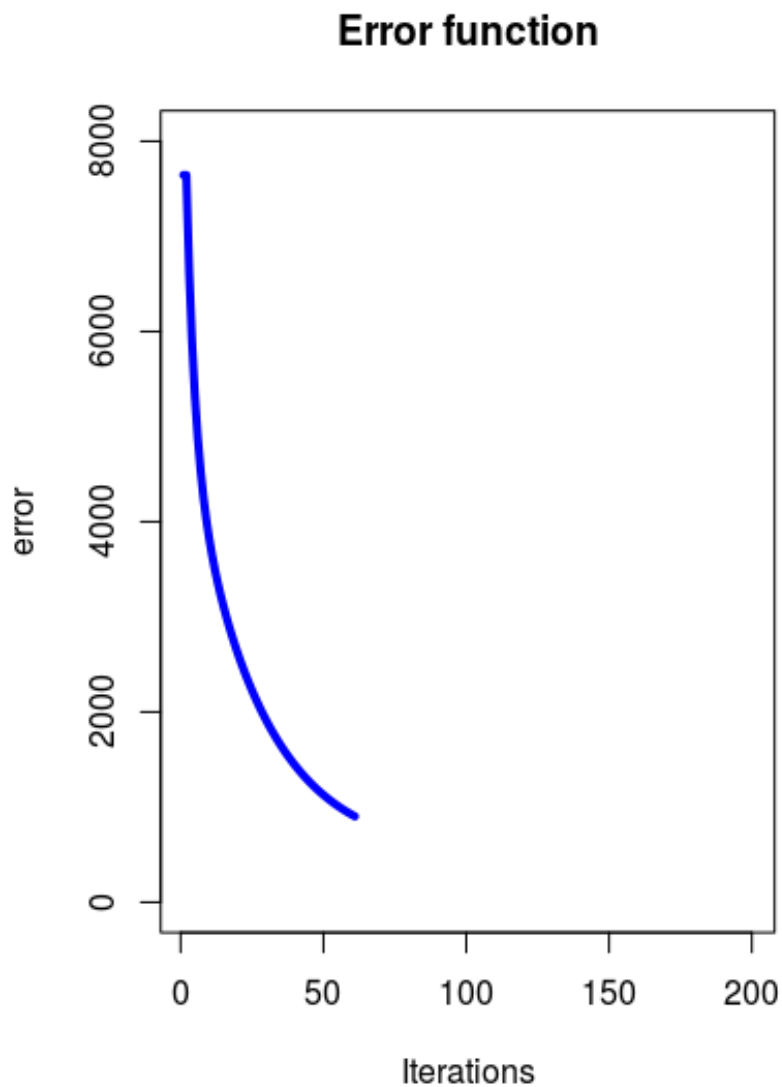
# Linear Regression - Animation



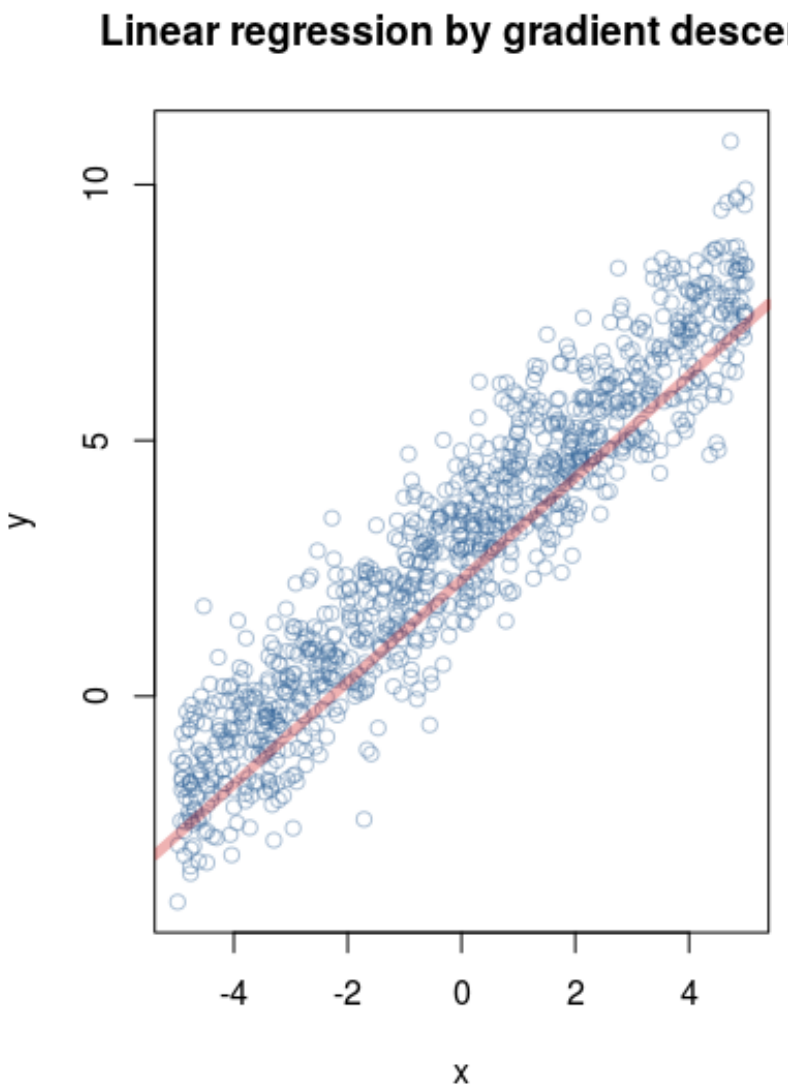



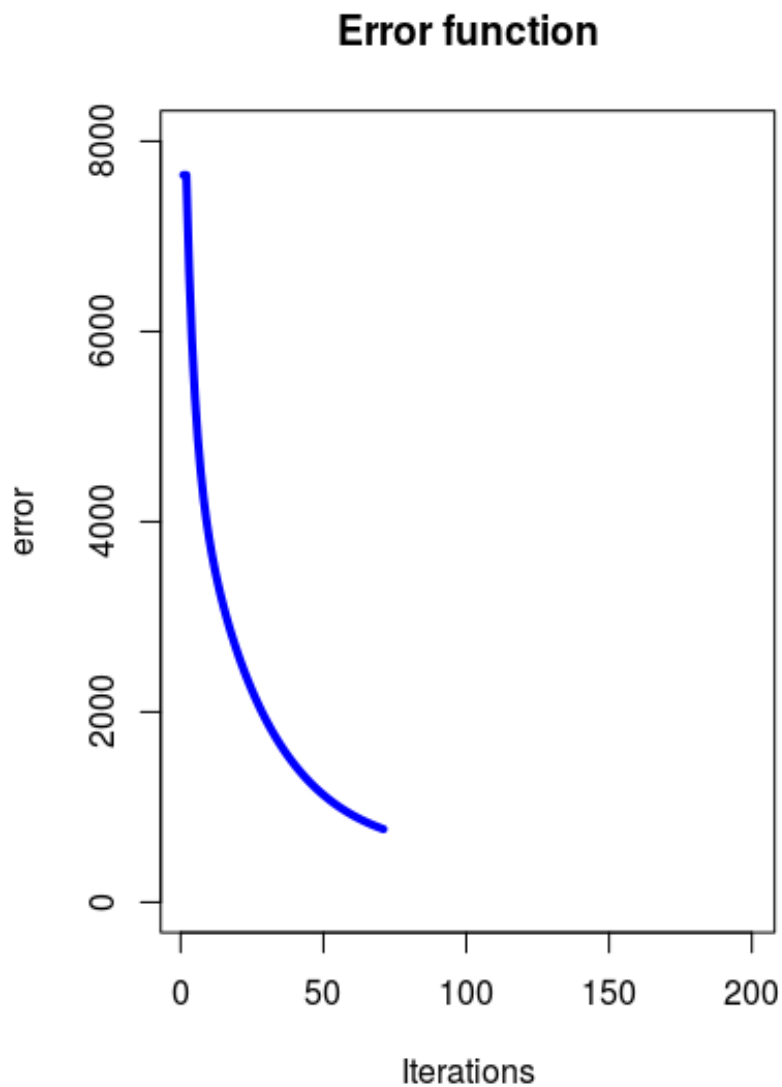
# Linear Regression - Animation



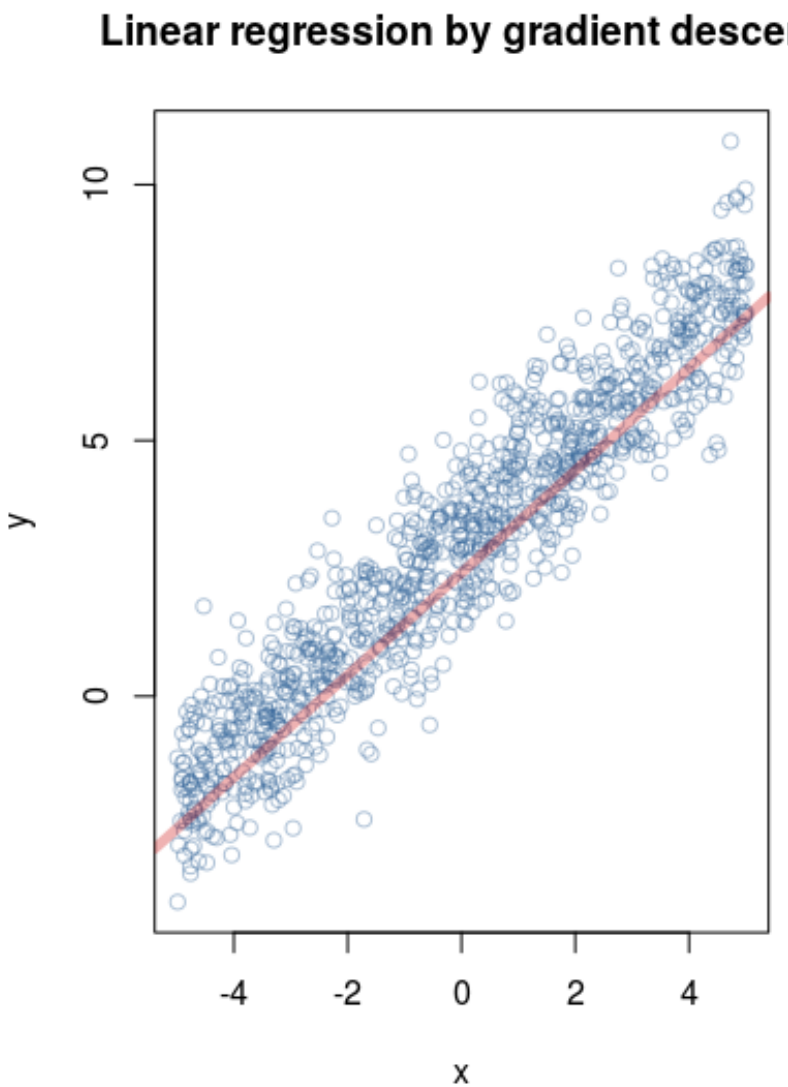



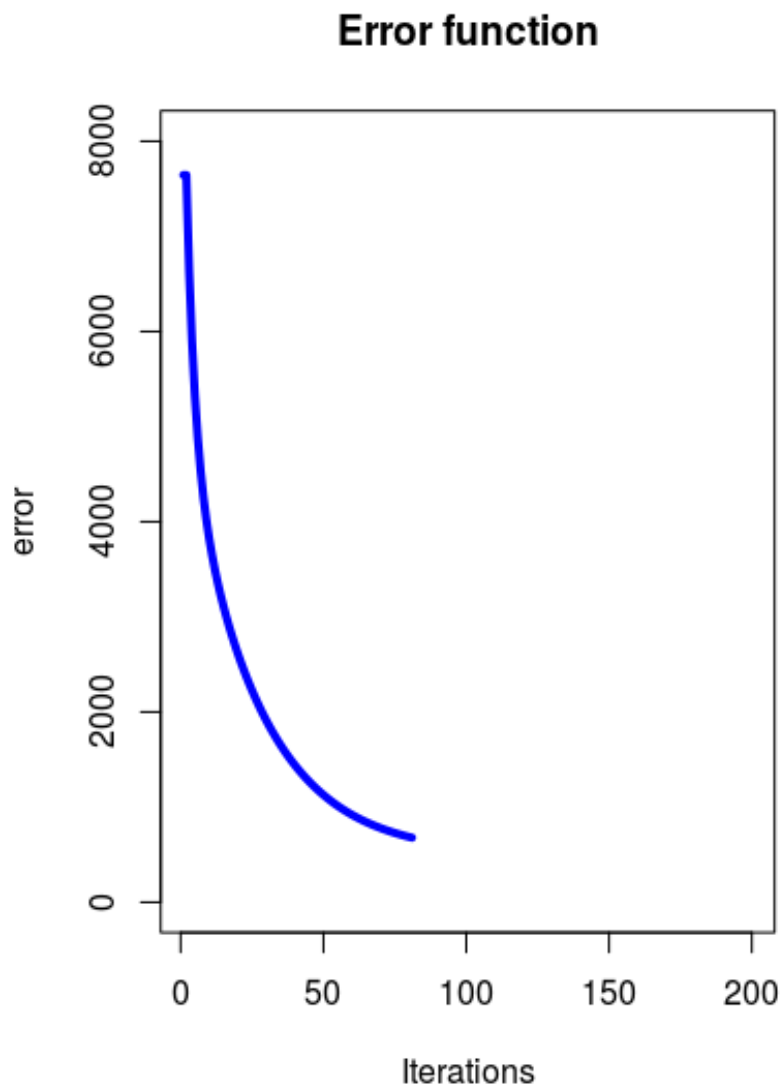
# Linear Regression - Animation



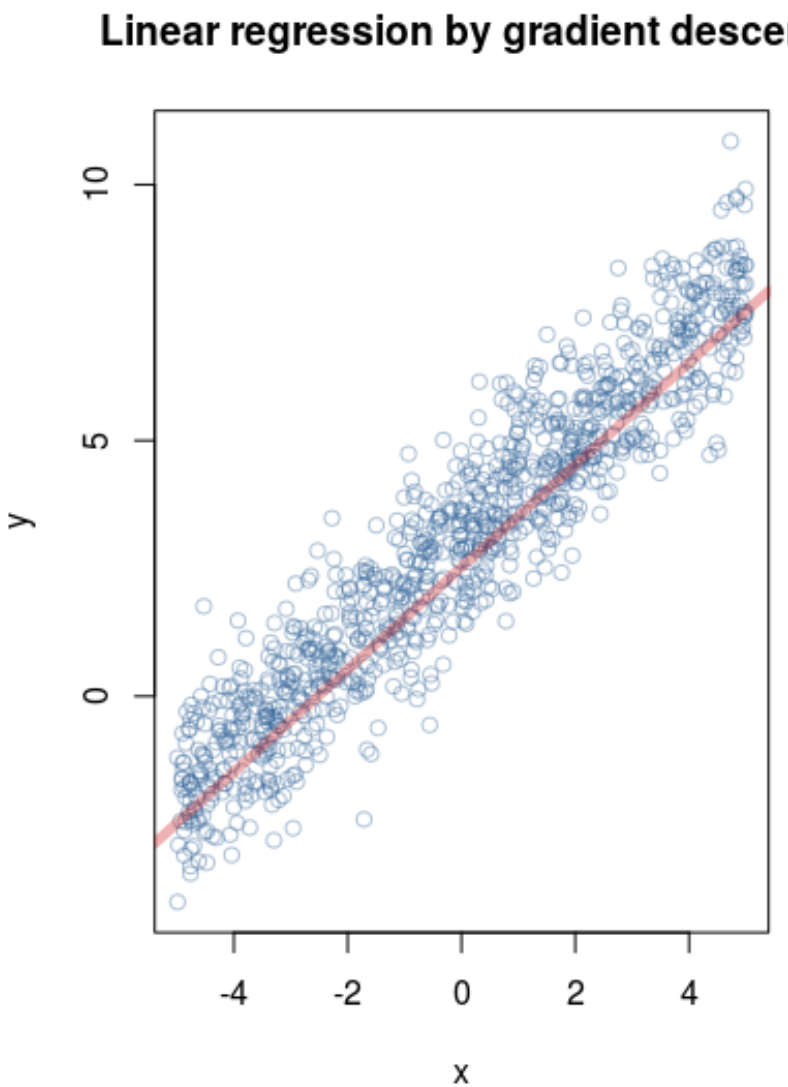



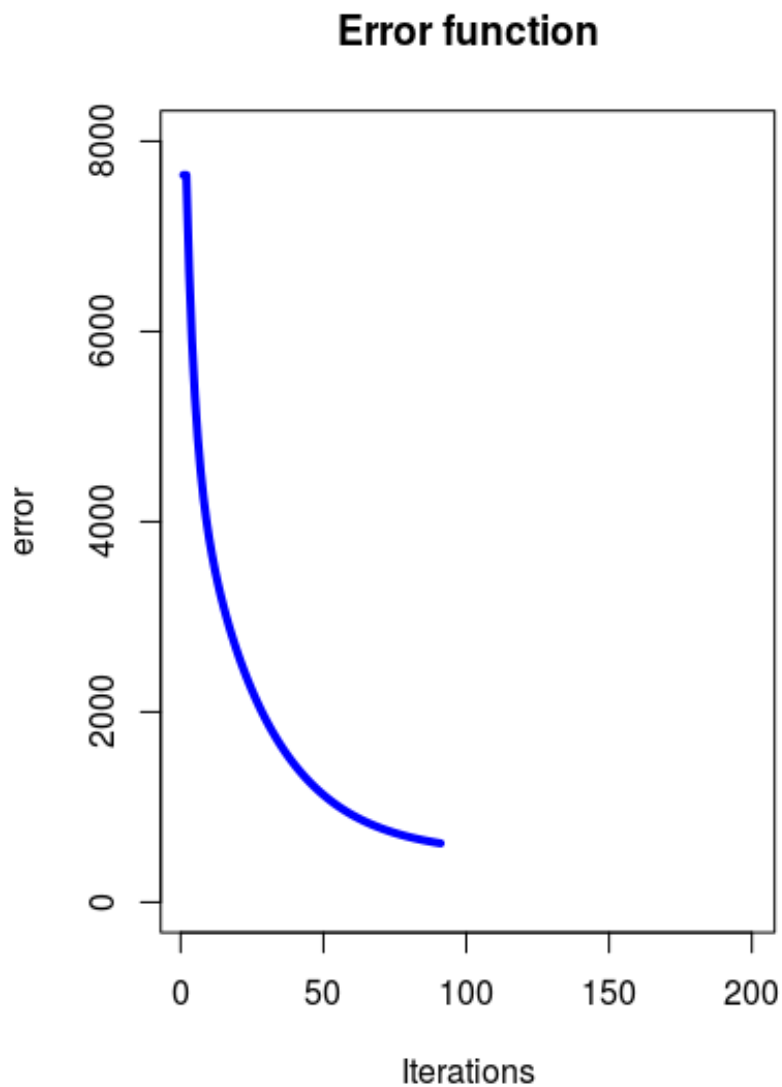
# Linear Regression - Animation



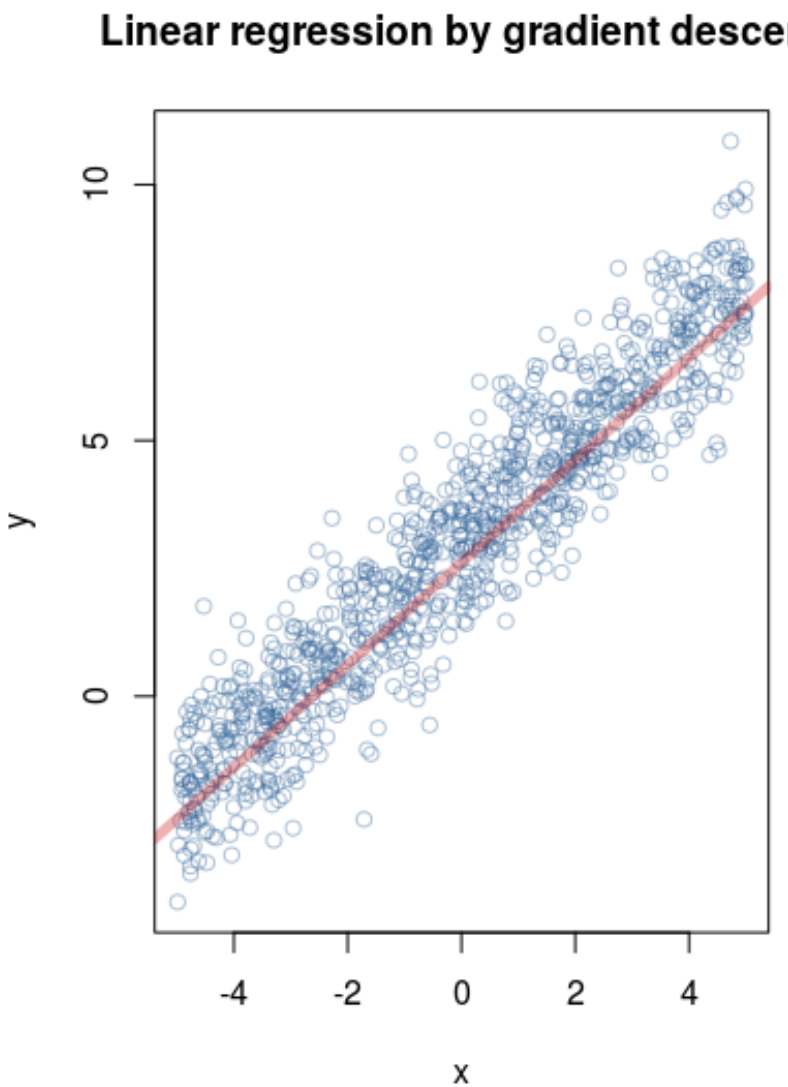



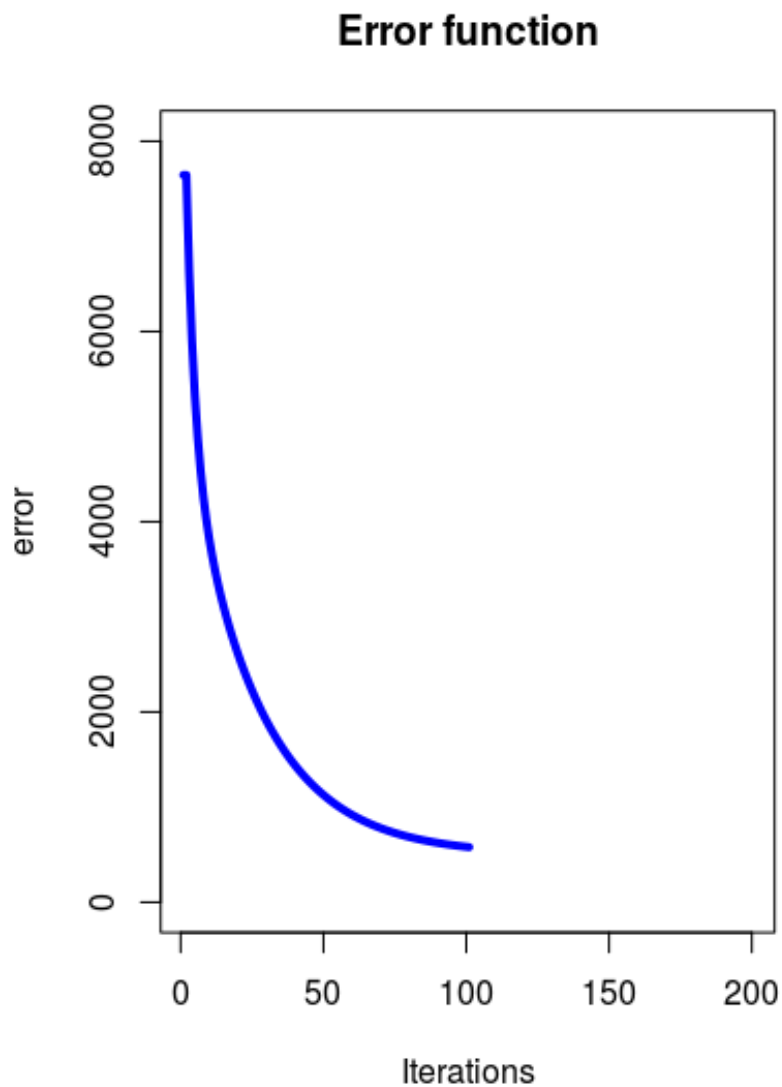
# Linear Regression - Animation



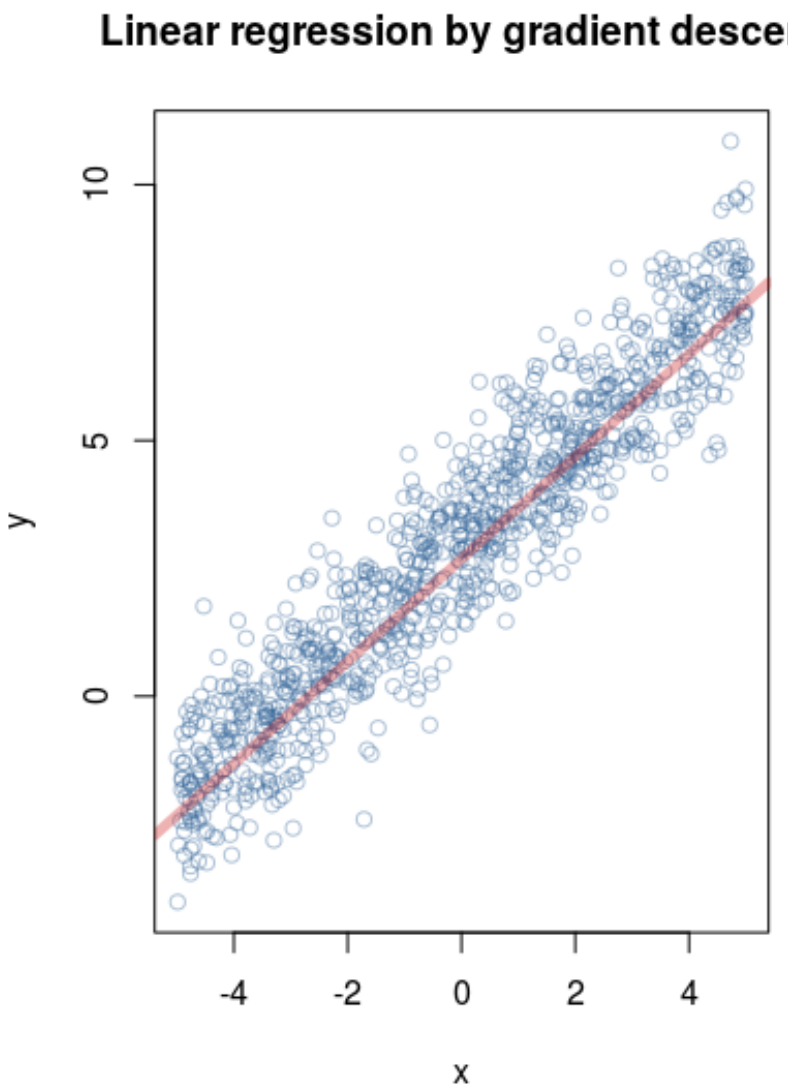



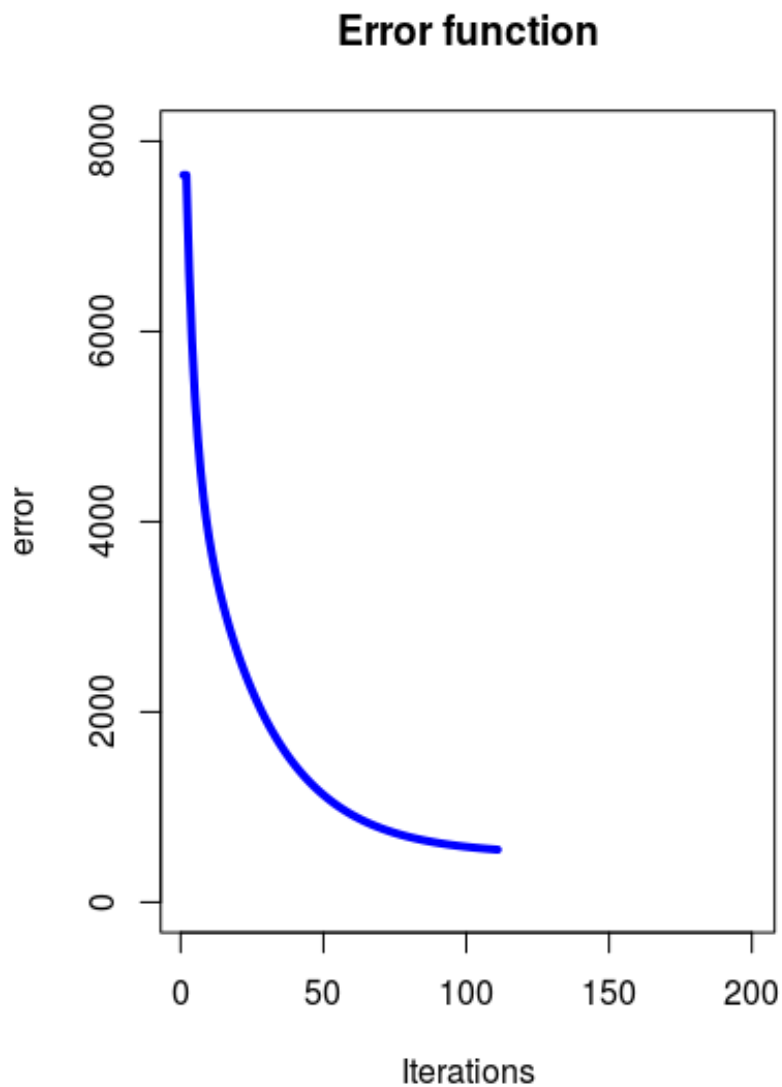
# Linear Regression - Animation



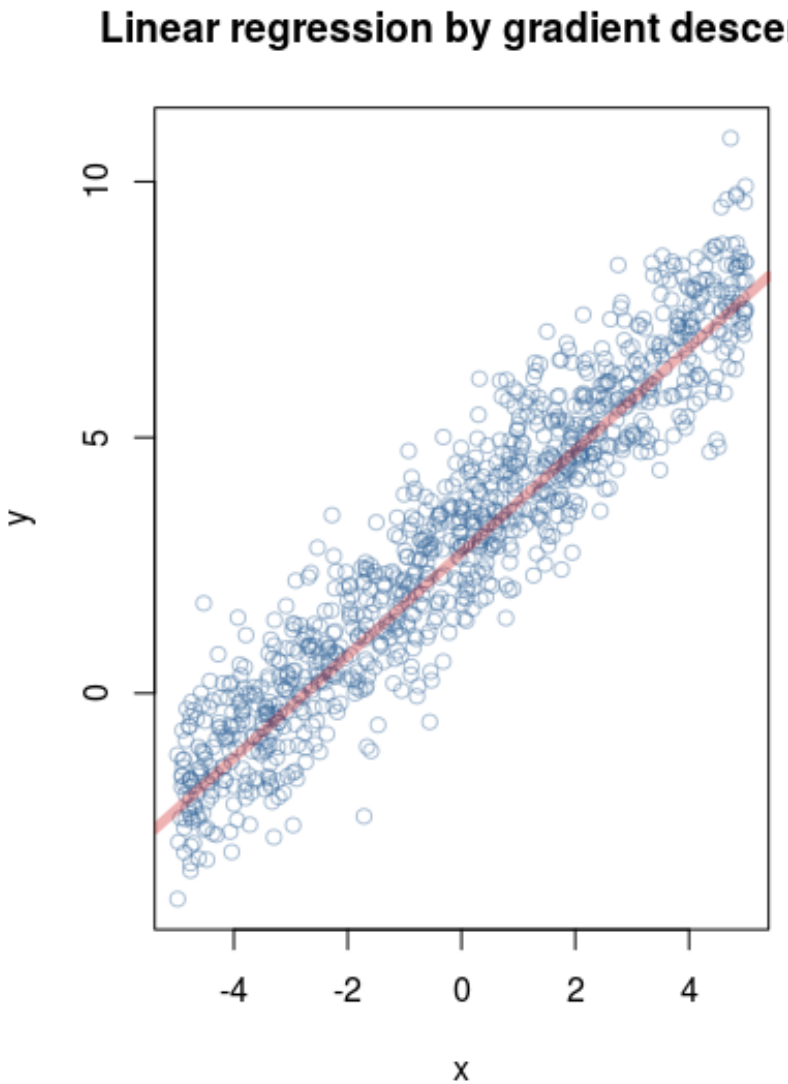



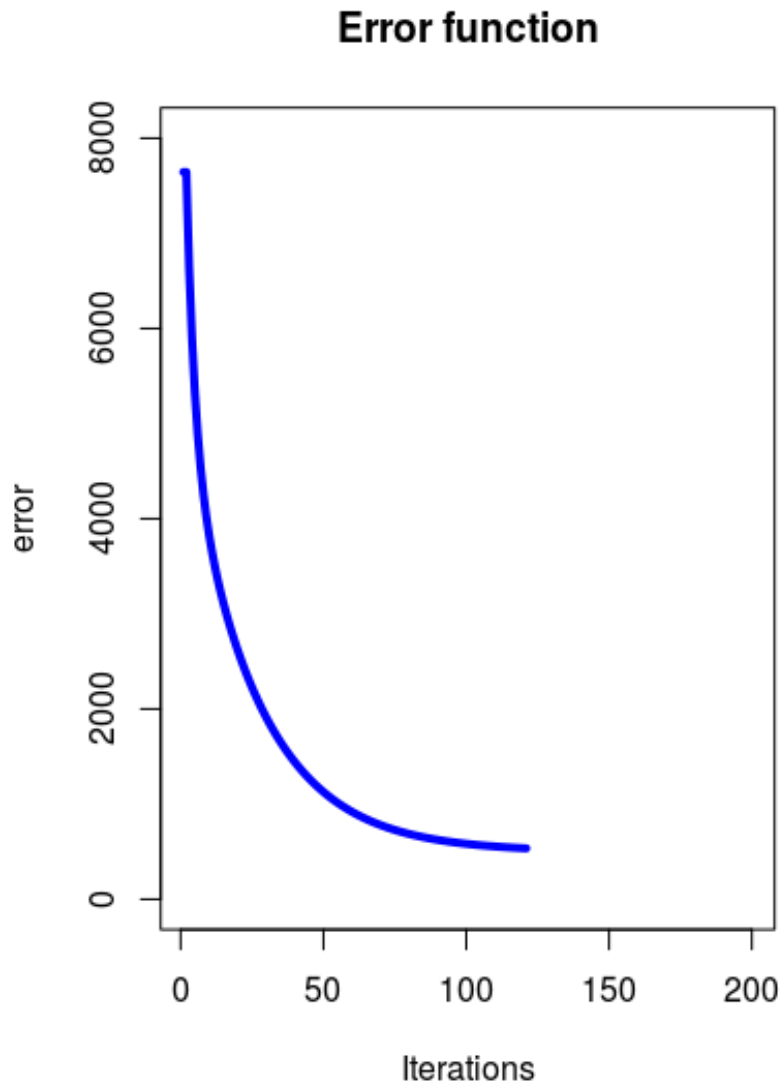
# Linear Regression - Animation



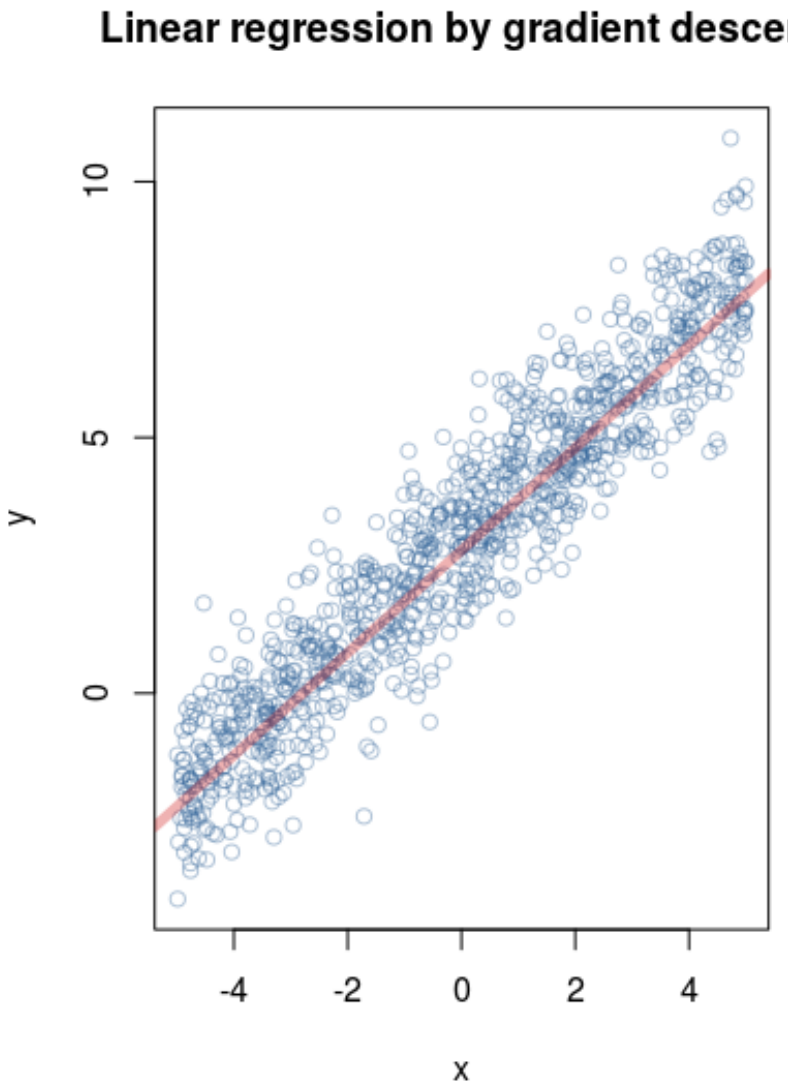



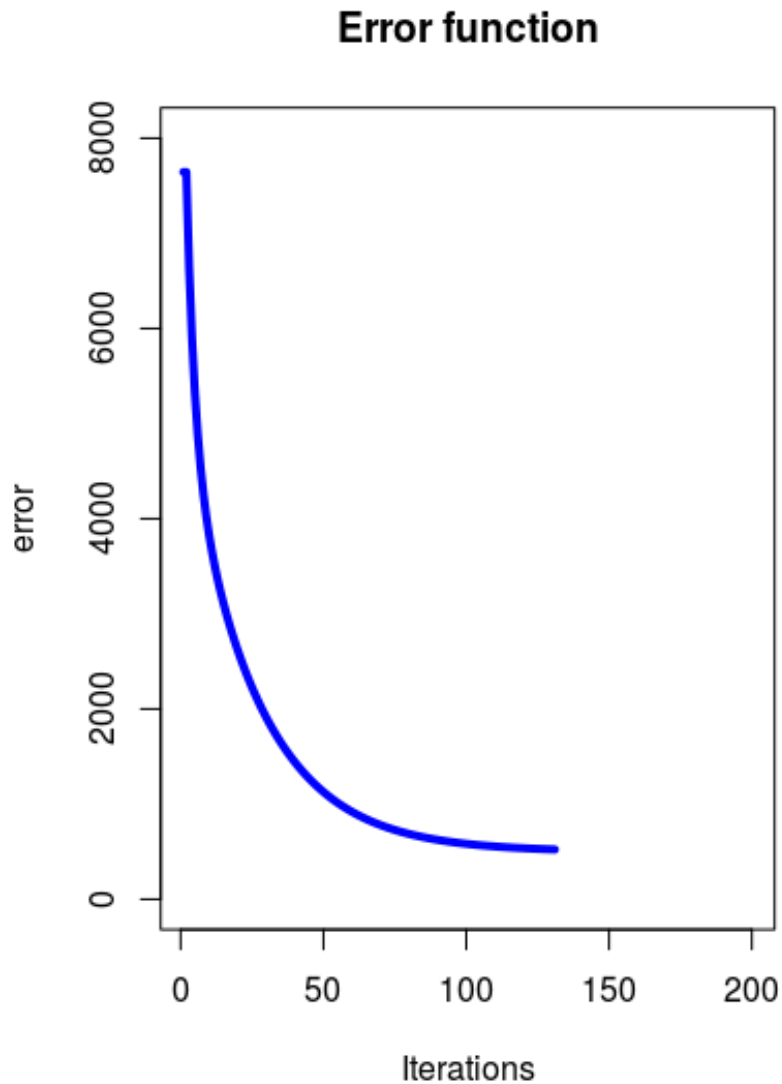
# Linear Regression - Animation



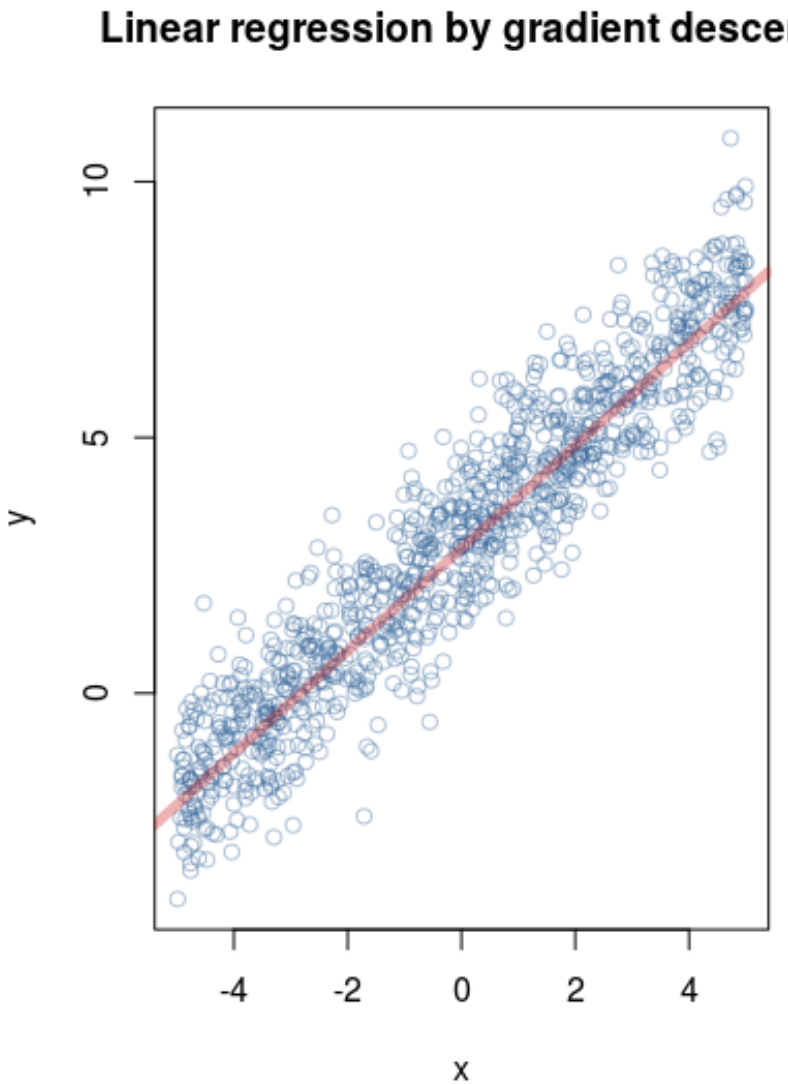



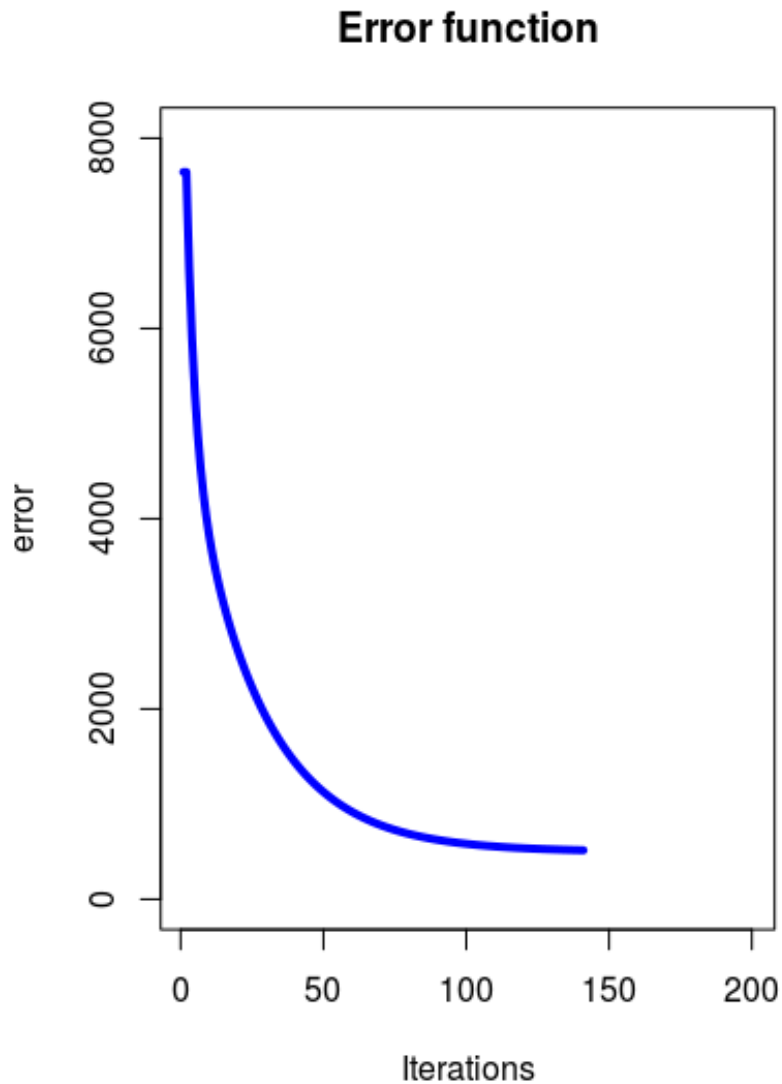
# Linear Regression - Animation



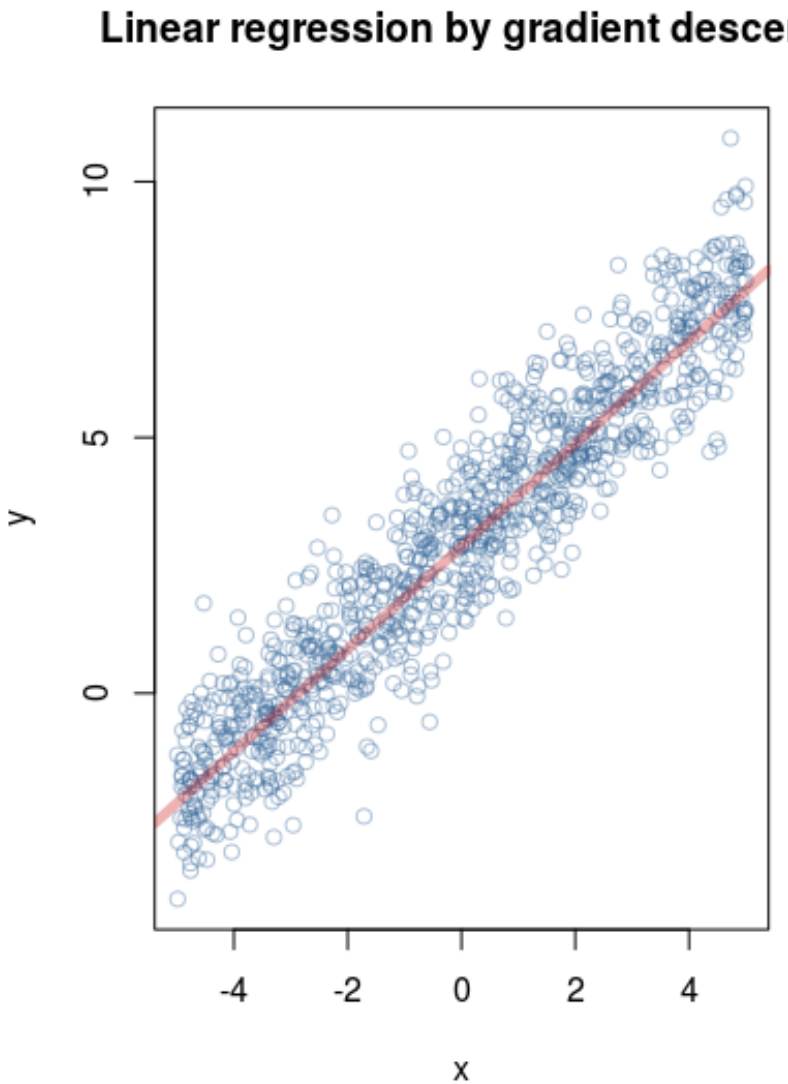



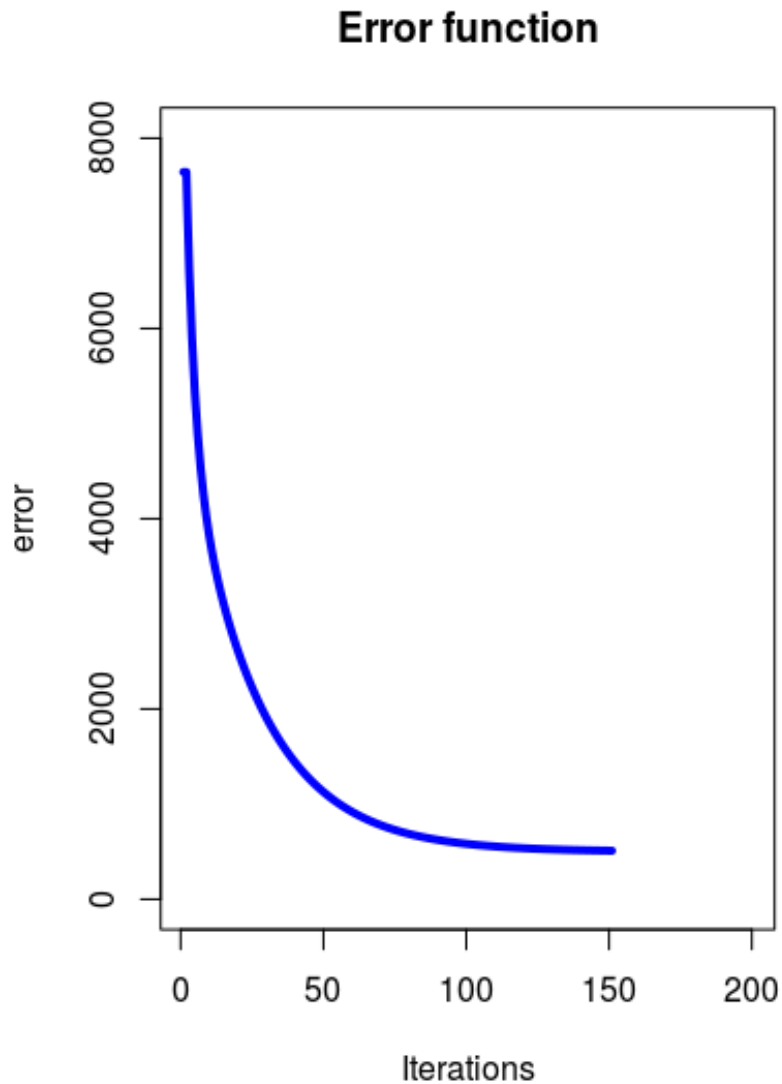
# Linear Regression - Animation



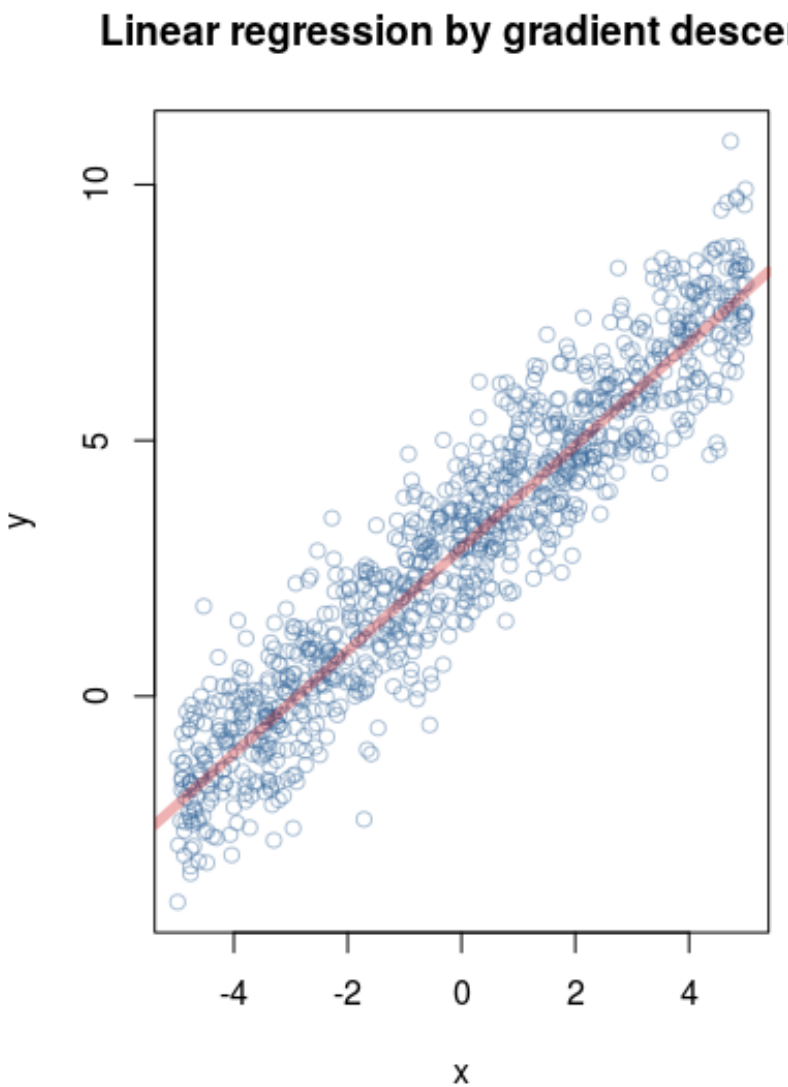



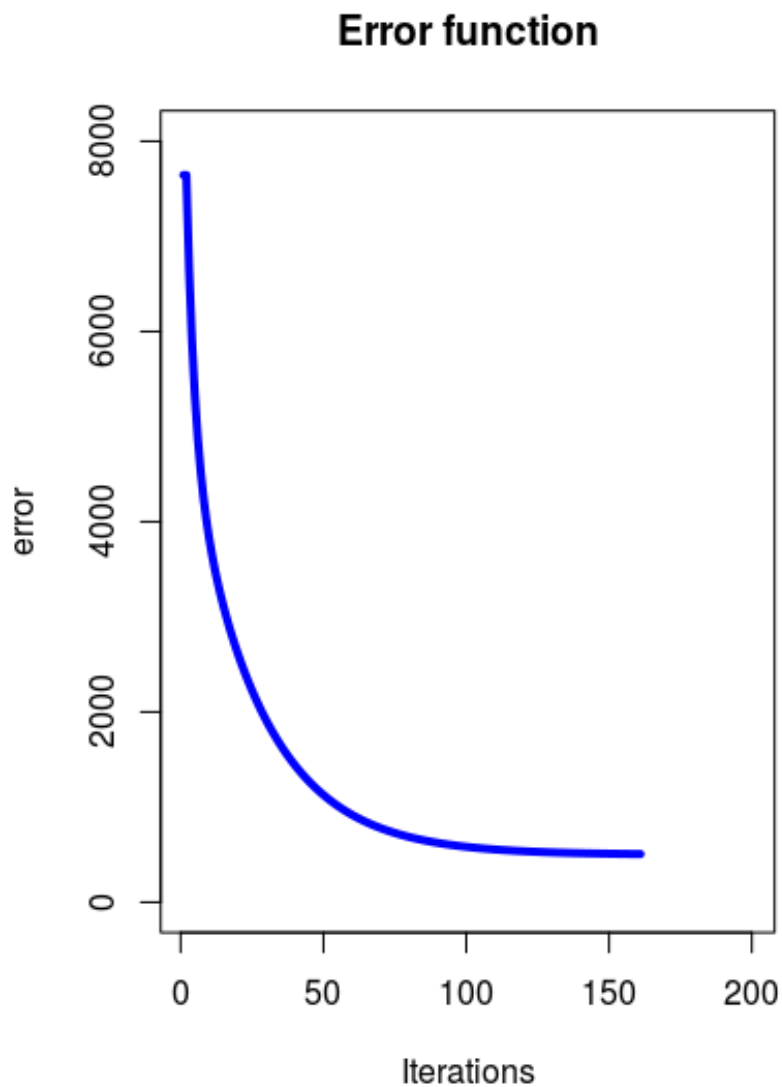
# Linear Regression - Animation



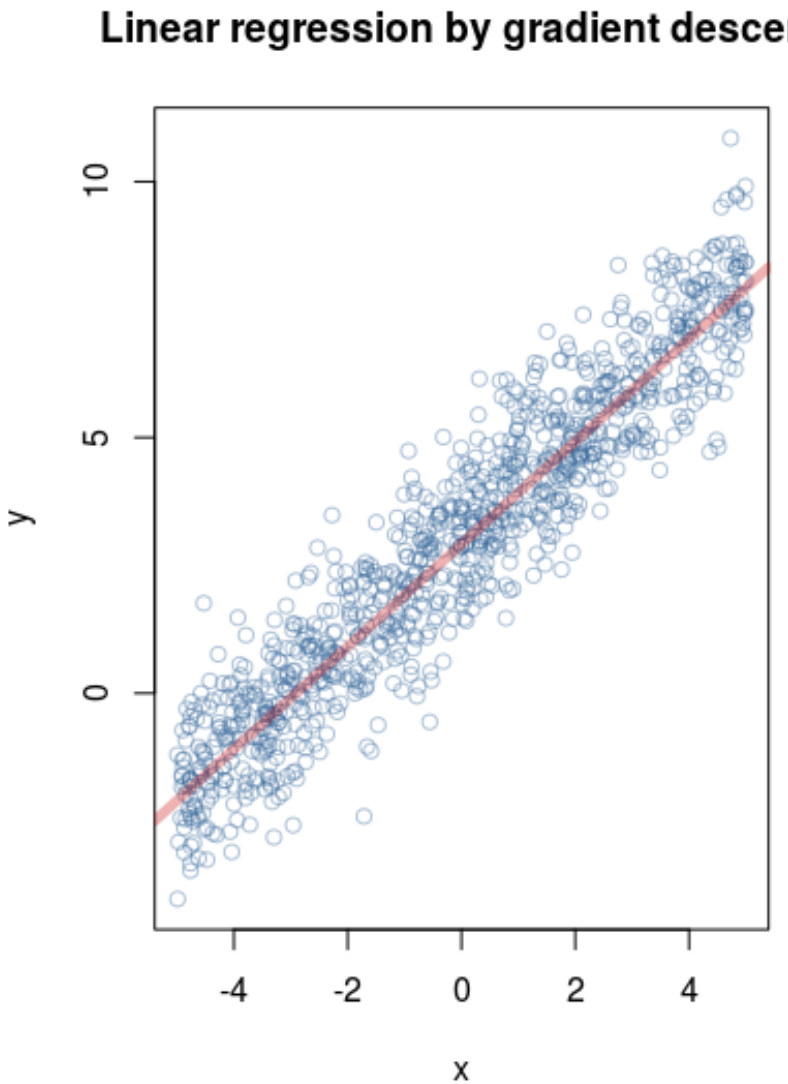



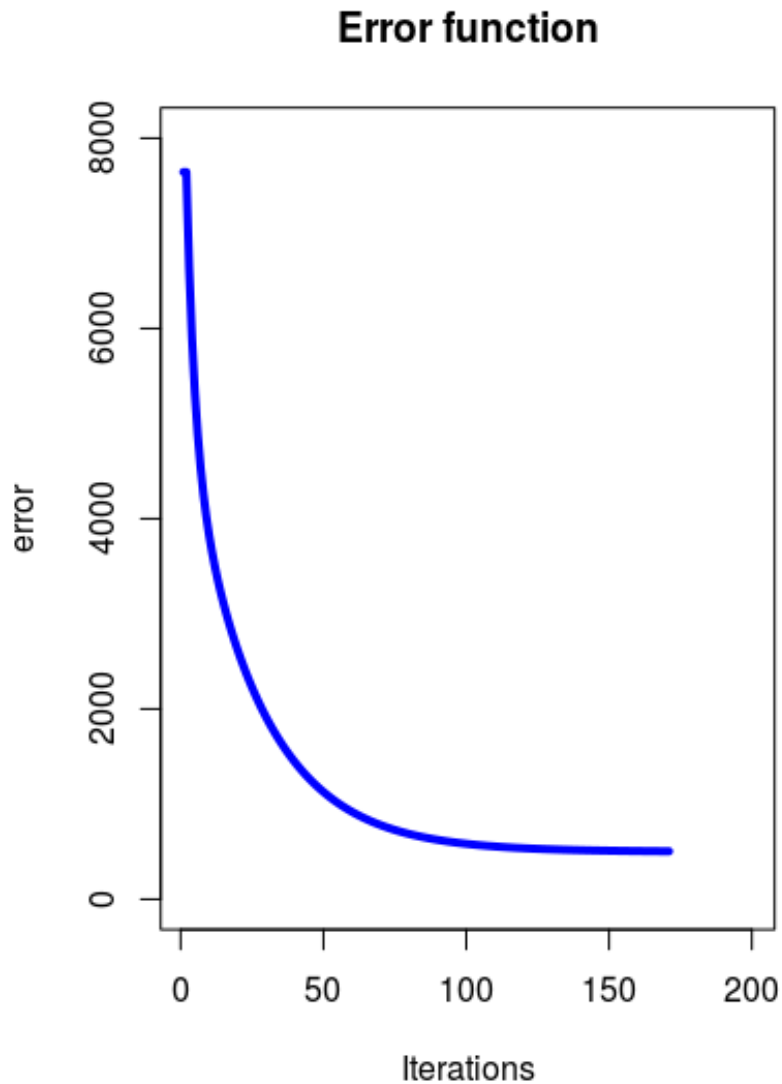
# Linear Regression - Animation



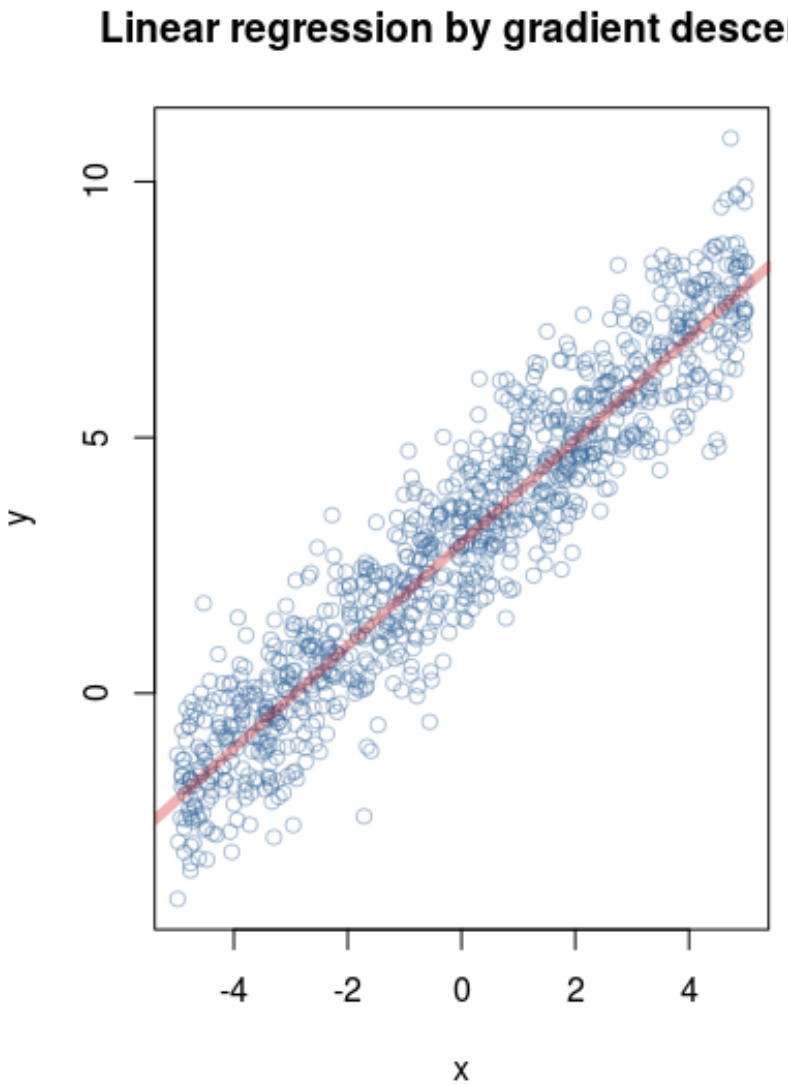



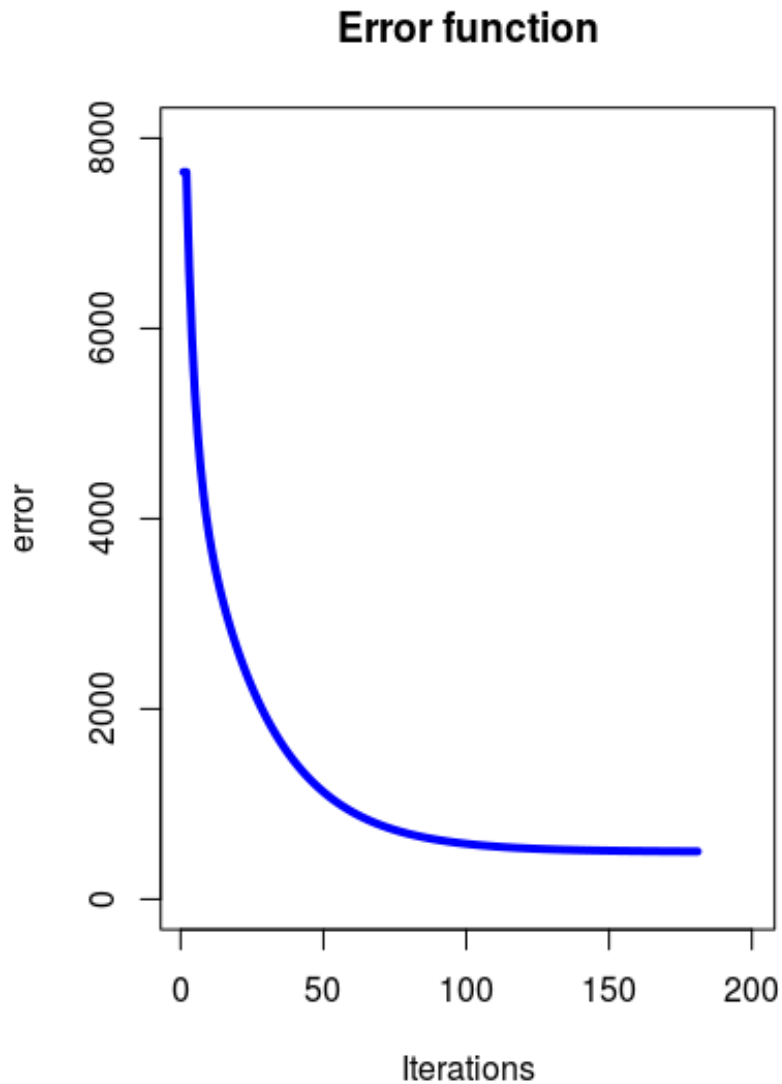
# Linear Regression - Animation



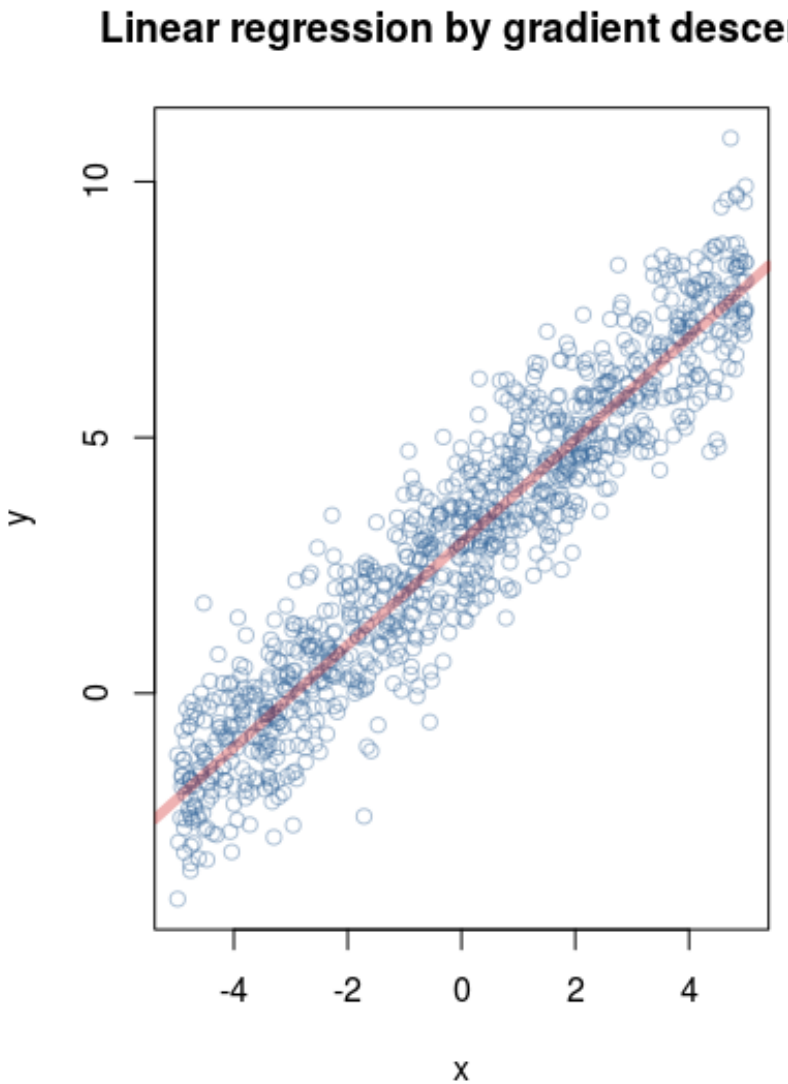



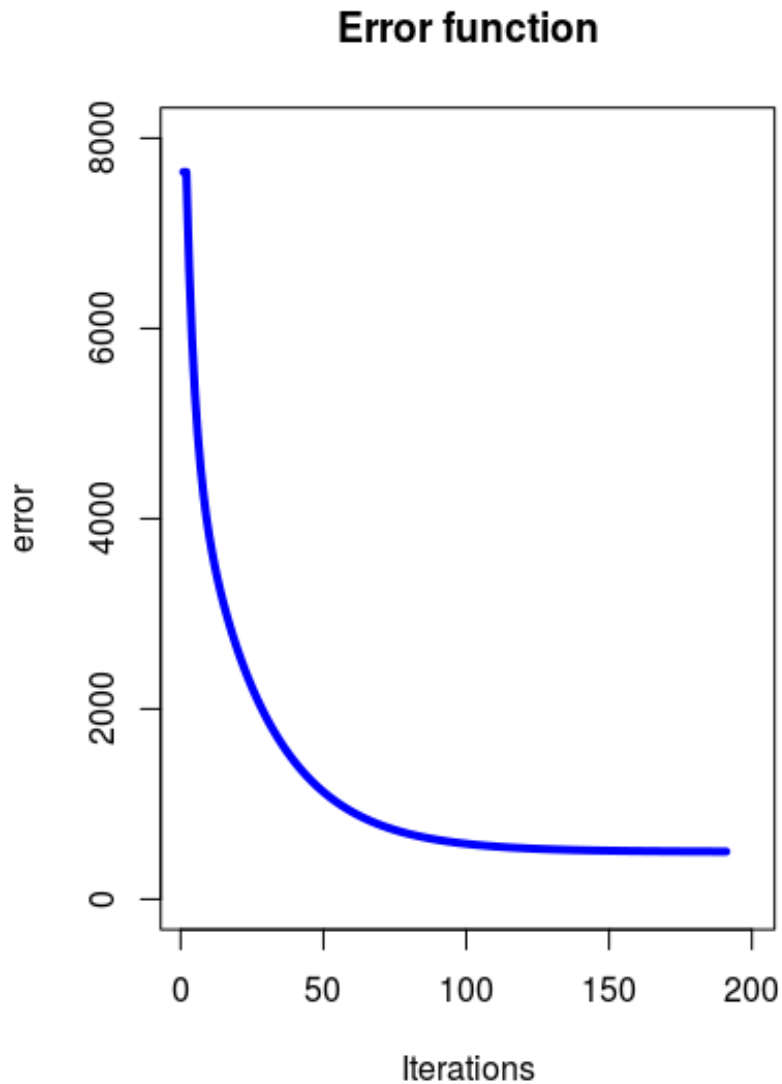
# Linear Regression - Animation



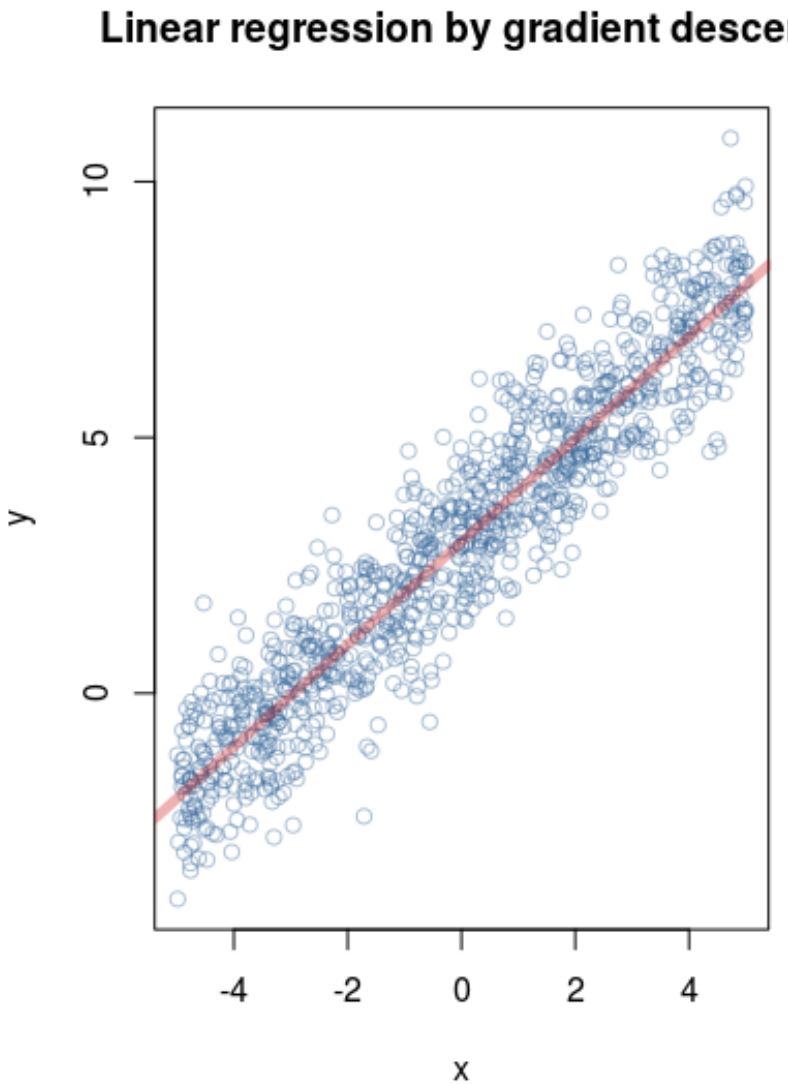



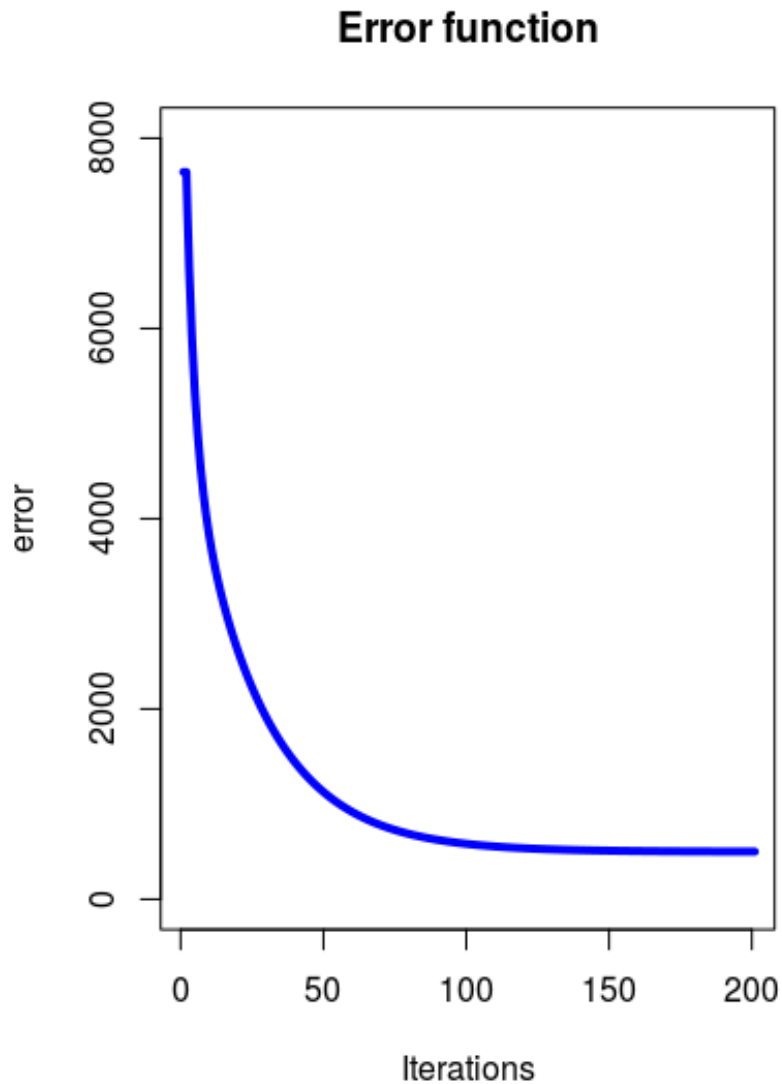
# Linear Regression - Animation



# Linear Regression - Animation



# Finding the Minimum in Dimension One

Assume $n = 1$. Then, the error function $E$ is

$$E(w_0, w_1) = \frac{1}{2} \sum_{k=1}^{p} (w_0 + w_1 x_k - f_k)^2$$

Minimize $E$ w.r.t. $w_0$ a $w_1$:

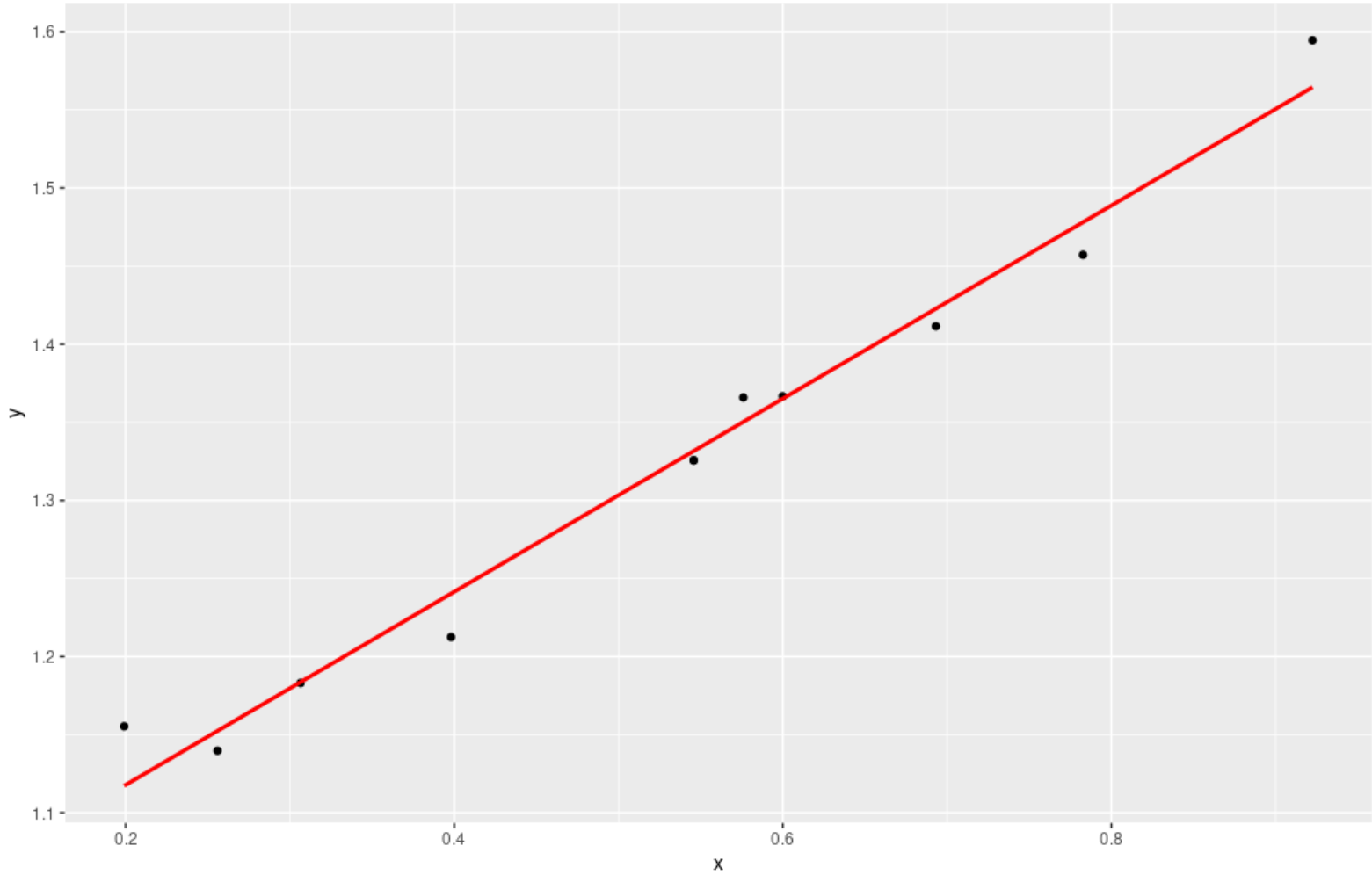
$$\frac{\partial E}{\partial w_0} = 0 \quad \Leftrightarrow \quad w_0 = \bar{f} - w_1 \bar{x} \quad \Leftrightarrow \quad \bar{f} = w_0 + w_1 \bar{x}$$

where $\bar{x} = \frac{1}{p} \sum_{k=1}^{p} x_k$ a $\bar{f} = \frac{1}{p} \sum_{k=1}^{p} f_k$

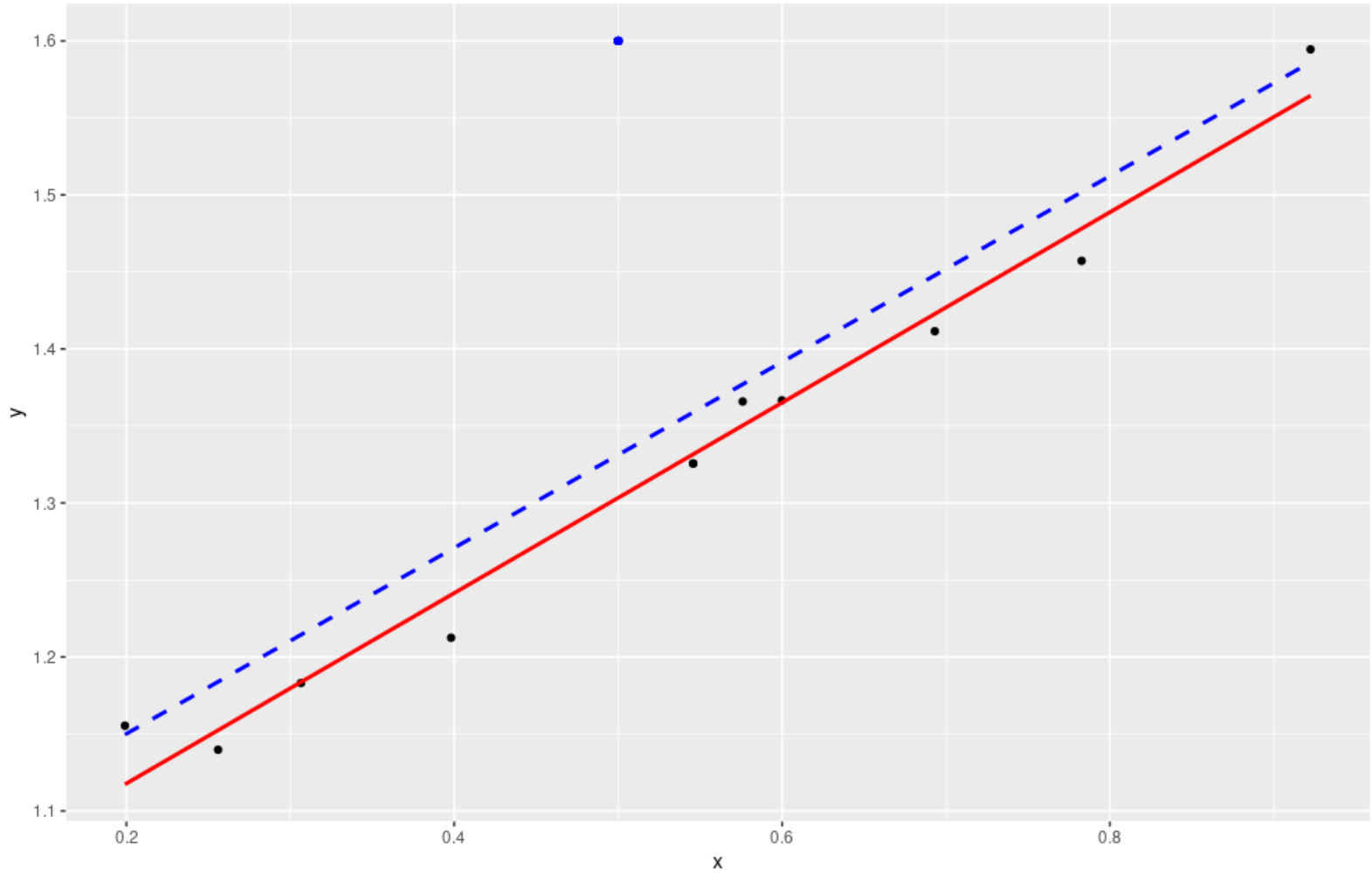
$$\frac{\partial E}{\partial w_1} = 0 \quad \Leftrightarrow \quad w_1 = \frac{\frac{1}{p} \sum_{k=1}^{p} (f_k - \bar{f})(x_k - \bar{x})}{\frac{1}{p} \sum_{k=1}^{p} (x_k - \bar{x})^2}$$

i.e. $w_1 = cov(f, x)/var(x)$

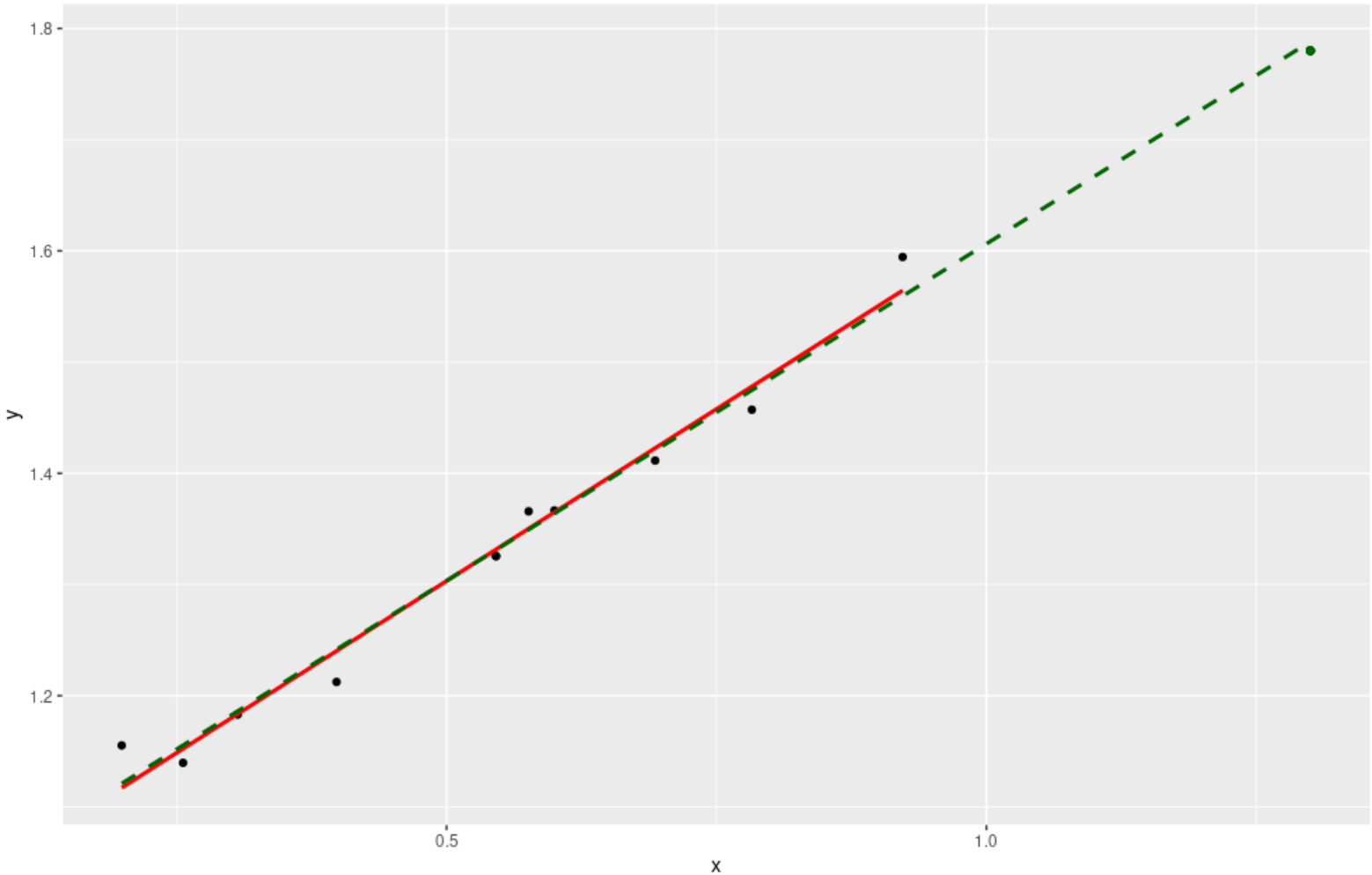
# Effect of Outliers



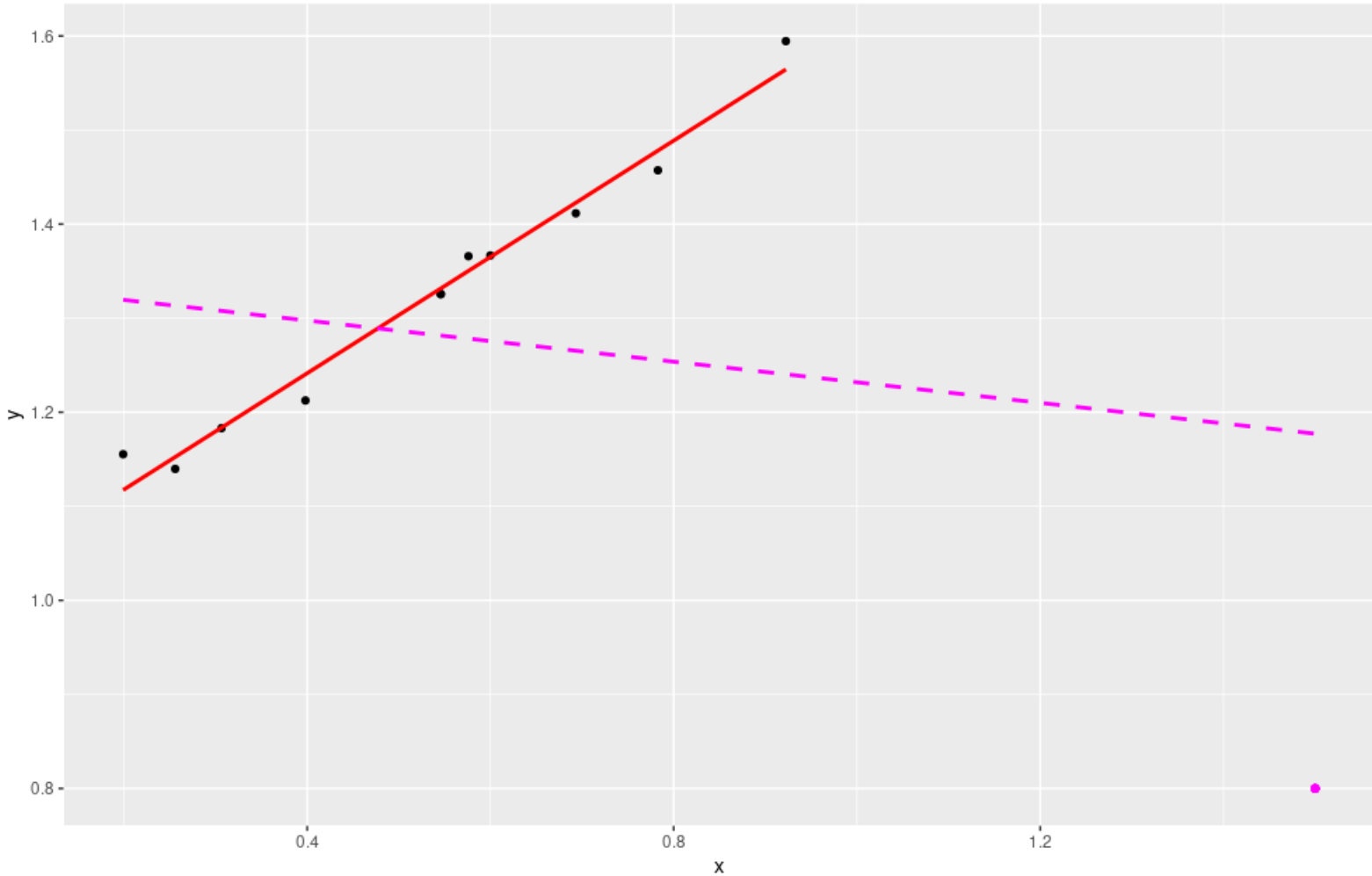
# Effect of Outliers



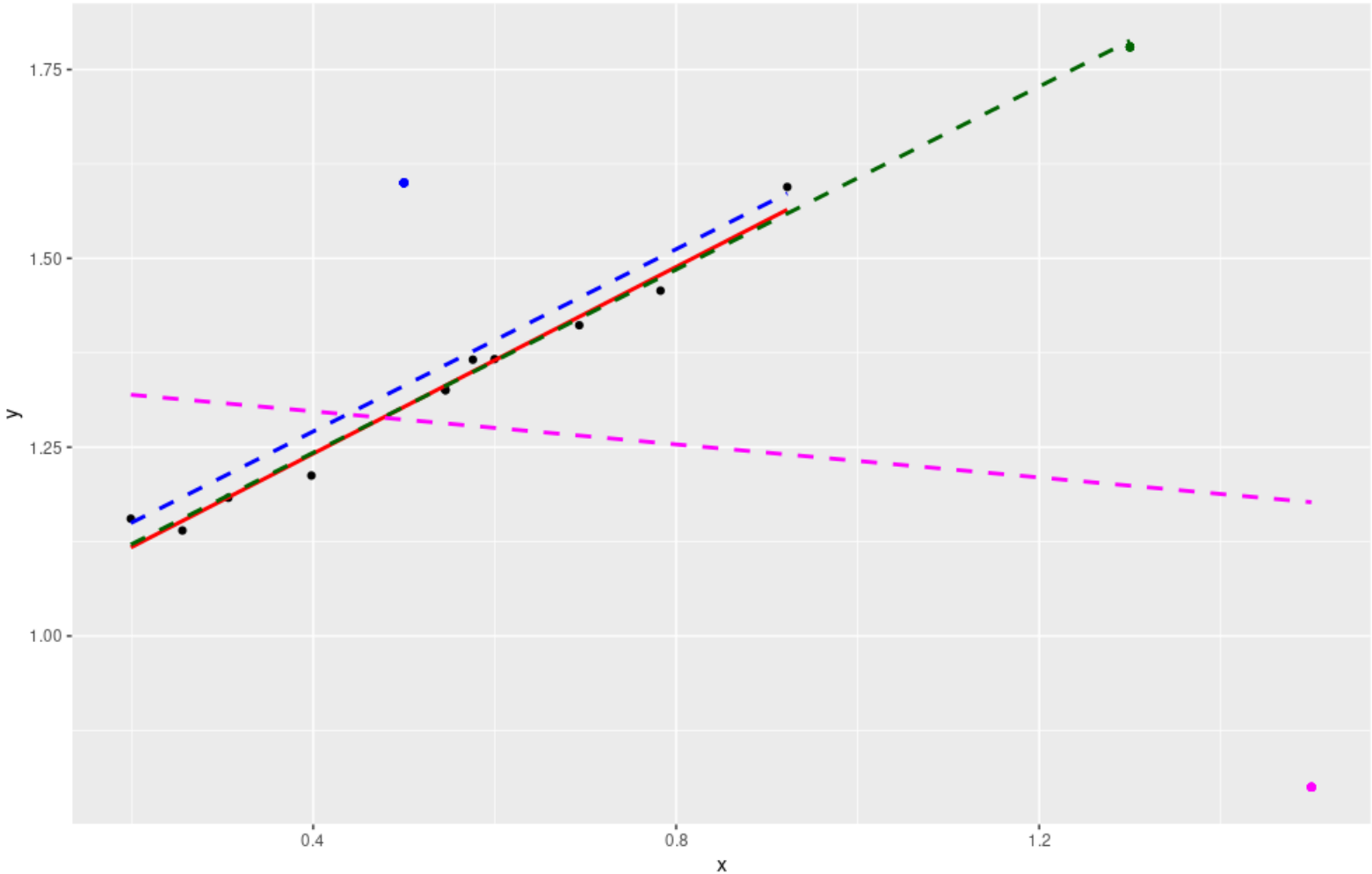
# Effect of Outliers



# Effect of Outliers



# Effect of Outliers

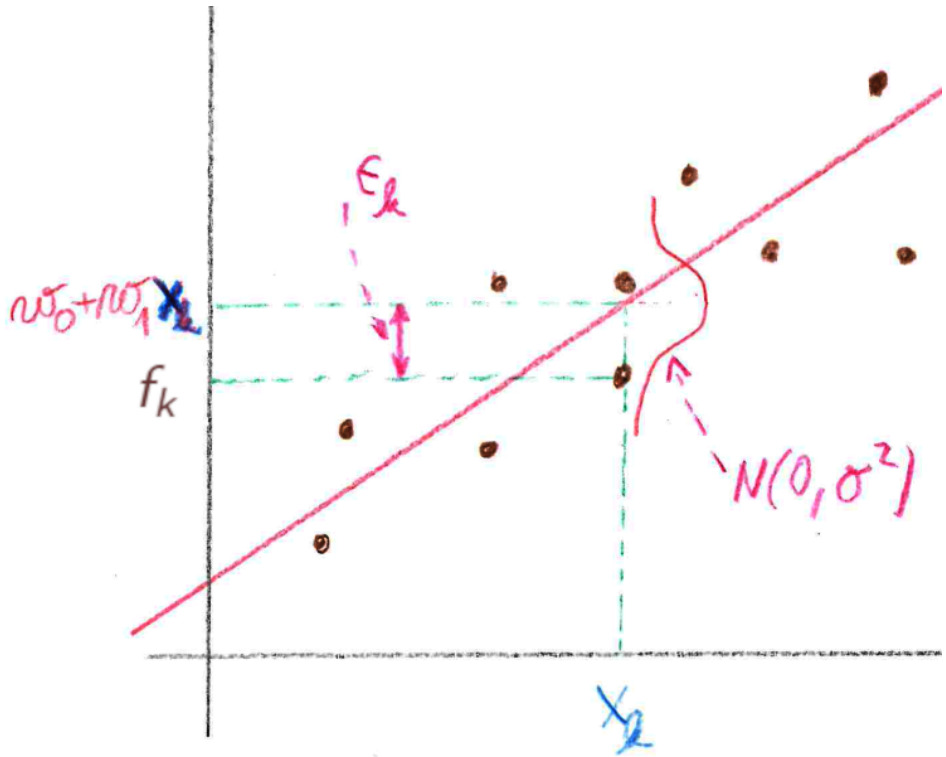


# Maximum Likelihood vs Least Squares (Dim 1)

**Fix a training set** $D = \{(x_1, f_1), (x_2, f_2), \ldots, (x_p, f_p)\}$
Assume that each $f_k$ has been generated randomly by

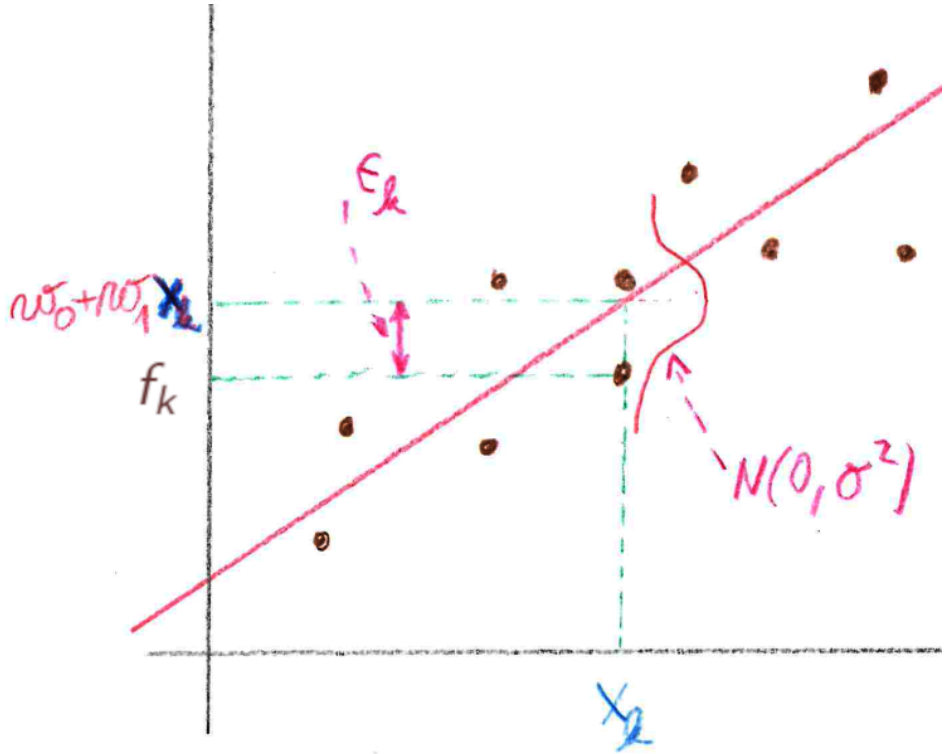
$$f_k = (w_0 + w_1 \cdot x_k) + \epsilon_k$$

where $w_0, w_1$ are **unknown weights**, and $\epsilon_k$ are independent, normally distributed noise values with mean 0 and some variance $\sigma^2$




How "probable" is it to generate the correct $f_1, \ldots, f_p$ ?

# Maximum Likelihood vs Least Squares (Dim 1)




How "probable" is it to generate the correct $f_1, \ldots, f_p$ ?

The following conditions are equivalent:

- $w_0, w_1$ minimize the squared error $E$
- $w_0, w_1$ maximize the likelihood (i.e., the "probability") of generating the correct values $f_1, \ldots, f_p$ using $f_k = (w_0 + w_1 \cdot x_k) + \epsilon_k$

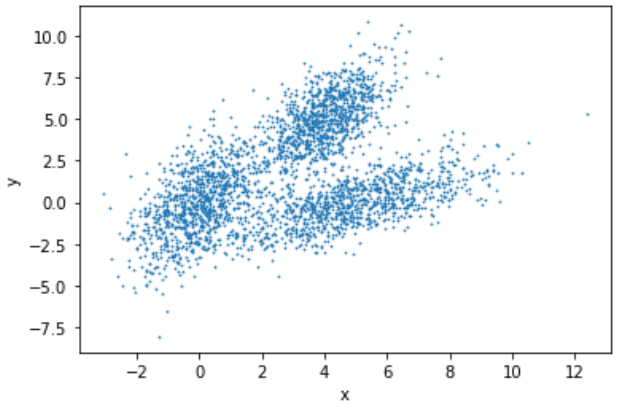
# Comments on Linear Models

- Linear models are parametric, i.e., they have a fixed form with a small number of parameters that need to be learned from data (as opposed, e.g. to decision trees where the structure is not fixed in advance).
- Linear models are stable, i.e., small variations in the training data have only limited impact on the learned model. (tree models typically vary more with the training data).
- Linear models are less likely to overfit (low variance) the training data but sometimes tend to underfit (high bias).
- Linear models are prone to outliers.

# Unsupervised Learning

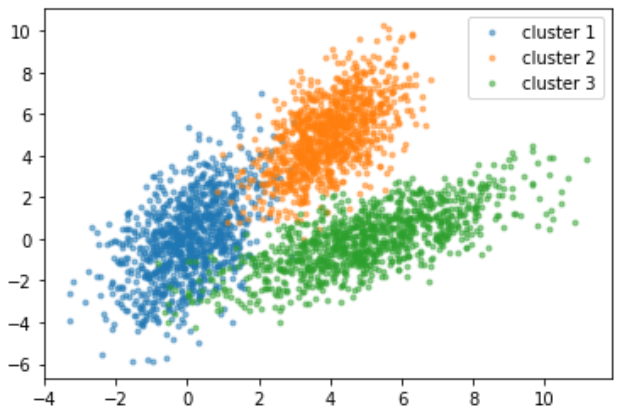
# Clustering

Often data form clusters based on some notion of similarity.




This means that the data distribution is *multimodal*, i.e., contains several regions of higher probability mass.

We aim to group data into clusters of "similar" examples without using any additional information. (no supervision).

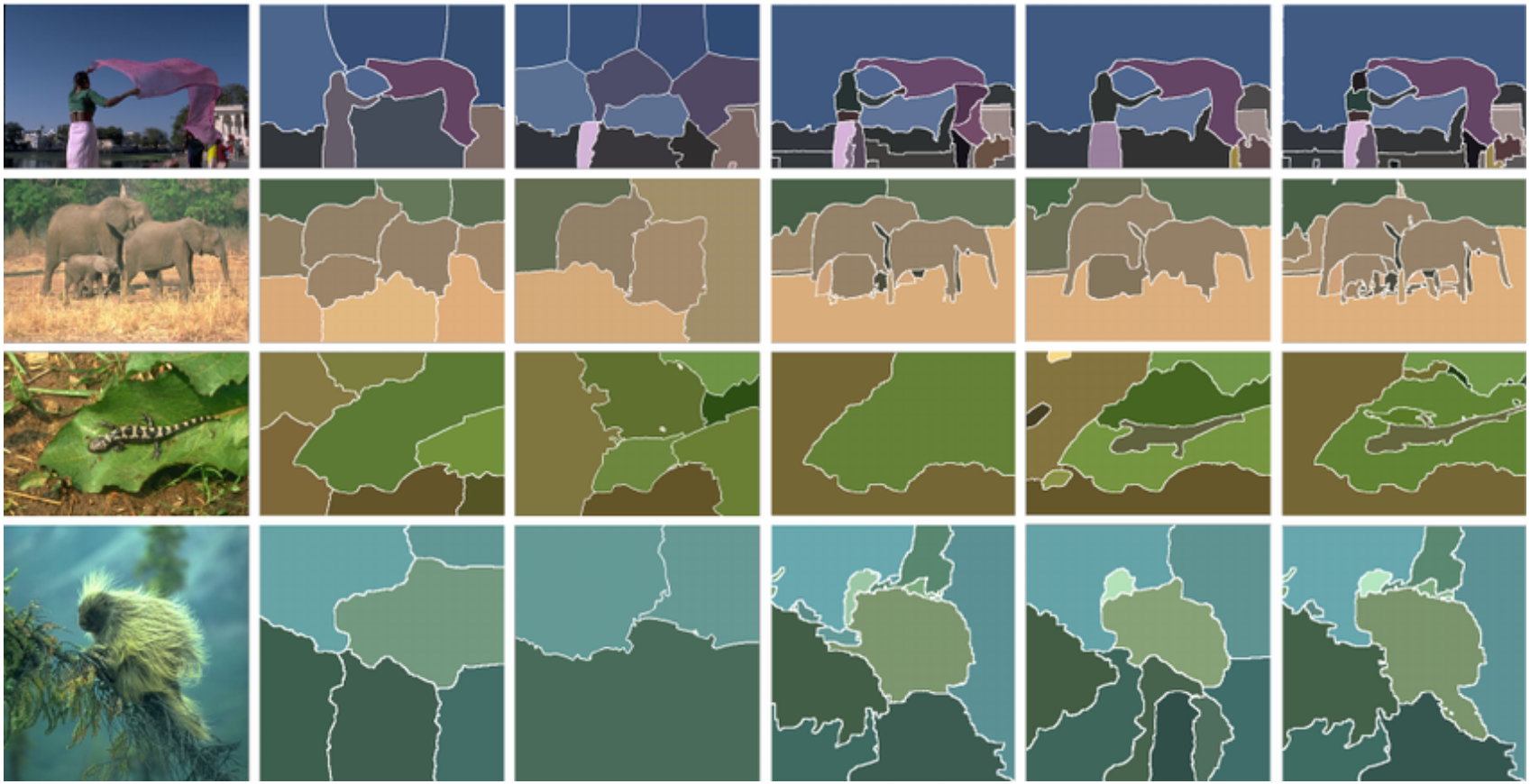


# Motivation

Clustering is useful, e.g., in

- Customer segmentation based on their purchases.
- Data exploration - identify patterns in data
- Semi-supervised learning - cluster labeled examples with the unlabeled ones
- Search engines - searching for images similar to a given image
- Image segmentation
- ...

# Segmentation



# Clustering Problem

Consider a dataset

$$D = \{\vec{x}_1, \ldots, \vec{x}_p\}$$

Note that no target class/value is provided.

*Clustering* is a partition $\mathcal{U} = \{U_1, \ldots, U_K\}$ of $D$ into $K$ clusters.

How do we identify the clusters?

Assume that we have a distance measure $d$ measuring how far apart the objects being clustered. We want close objects to be clustered together.

**For concreteness**:

- We stick with numerical features, which means that the dataset $D = \{\vec{x}_1, \ldots, \vec{x}_p\}$ contains vectors $\vec{x}_i \in \mathbb{R}^n$.
- Assume the Euclidean distance $d$.

Note that clustering may be based on completely different similarity/dissimilarity measures and non-numerical data.

# K-Means Clustering

# K-means clustering

The $K$-means clustering model consists of

- The number of clusters $K$
- $K$ cluster *prototypes* $\vec{m}_1, \ldots, \vec{m}_K \in \mathbb{R}^n$
- An *assignment* $q_{ij} \in \{0, 1\}$ for $i = 1, \ldots, p$ and $j = 1, \ldots, K$ of inputs $\vec{x}_i$ to clusters $U_j$ so that

$$\sum_j q_{ij} = 1 \qquad \text{for } i = 1, \ldots, p$$

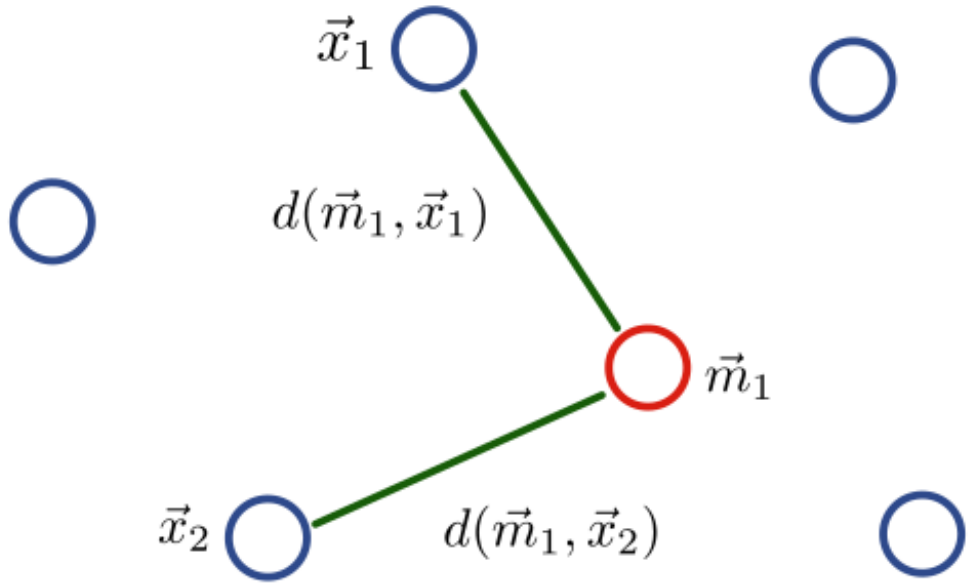
A given assignment $\{q_{ij}\}$ induces a clustering

$$\mathcal{U} = \{U_1, \ldots, U_K\} \text{ where } \vec{x}_i \in U_j \text{ iff } q_{ij} = 1$$

How good is a given model?

# Error Function

Measure the distance of inputs $\vec{x}_i$ to their cluster prototypes $\vec{m}_j$




$$E(\{q_{ij}\}, \{\vec{m}_j\}) = \sum_{i=1}^{p} \sum_{j=1}^{K} q_{ij} d(\vec{x}_i, \vec{m}_j)^2$$

We aim to *minimize* this error, i.e., to find proper positions of cluster prototypes and their assignment to minimize the total squared distance of examples to their prototypes.

# K-means Clustering Algorithm

**The Problem:** Minimize

$$E(\{q_{ij}\},\{\vec{m}_j\}) = \sum_{i=1}^{p}\sum_{j=1}^{K} q_{ij}\, d(\vec{x}_i,\vec{m}_j)^2 \text{ w.r.t. } \{q_{ij}\},\{\vec{m}_j\}$$

Note that

- If we fix $\{\vec{m}_j\}$, we can minimize $E(\{q_{ij}\},\{\vec{m}_j\})$ by setting $q_{ij} = 1$ iff $\vec{m}_j$ is the *closest prototype* to $\vec{x}_i$.
- If we fix $\{q_{ij}\}$, we can minimize $E(\{q_{ij}\},\{\vec{m}_j\})$ by letting each $\vec{m}_j$ to *minimize* the total squared distance to its prototypes:

  $$\sum_i q_{ij}\, d(\vec{x}_i,\vec{m}_j)^2$$

  This is achieved by putting each prototype $\vec{m}_j$ into the centroid of all inputs it represents:

  $$\vec{m}_j = \frac{1}{\sum_{i=1}^{p} q_{ij}} \sum_{i=1}^{p} q_{ij}\vec{x}_i$$

  Note that $\sum_{i=1}^{p} q_{ij}$ is the size of the cluster represented by $\vec{m}_j$.

# K-Means Clustering Algorithm

**Algorithm 1** K-means clustering

1: Initialize $K$ cluster centers $\vec{m}_1, \vec{m}_2, \ldots, \vec{m}_K$ randomly
2: **repeat**
3: **for** each data point $\vec{x}_i$ **do**
4: Assign $\vec{x}_i$ to the nearest centroid, i.e., set $q_{ij} = 1$ for

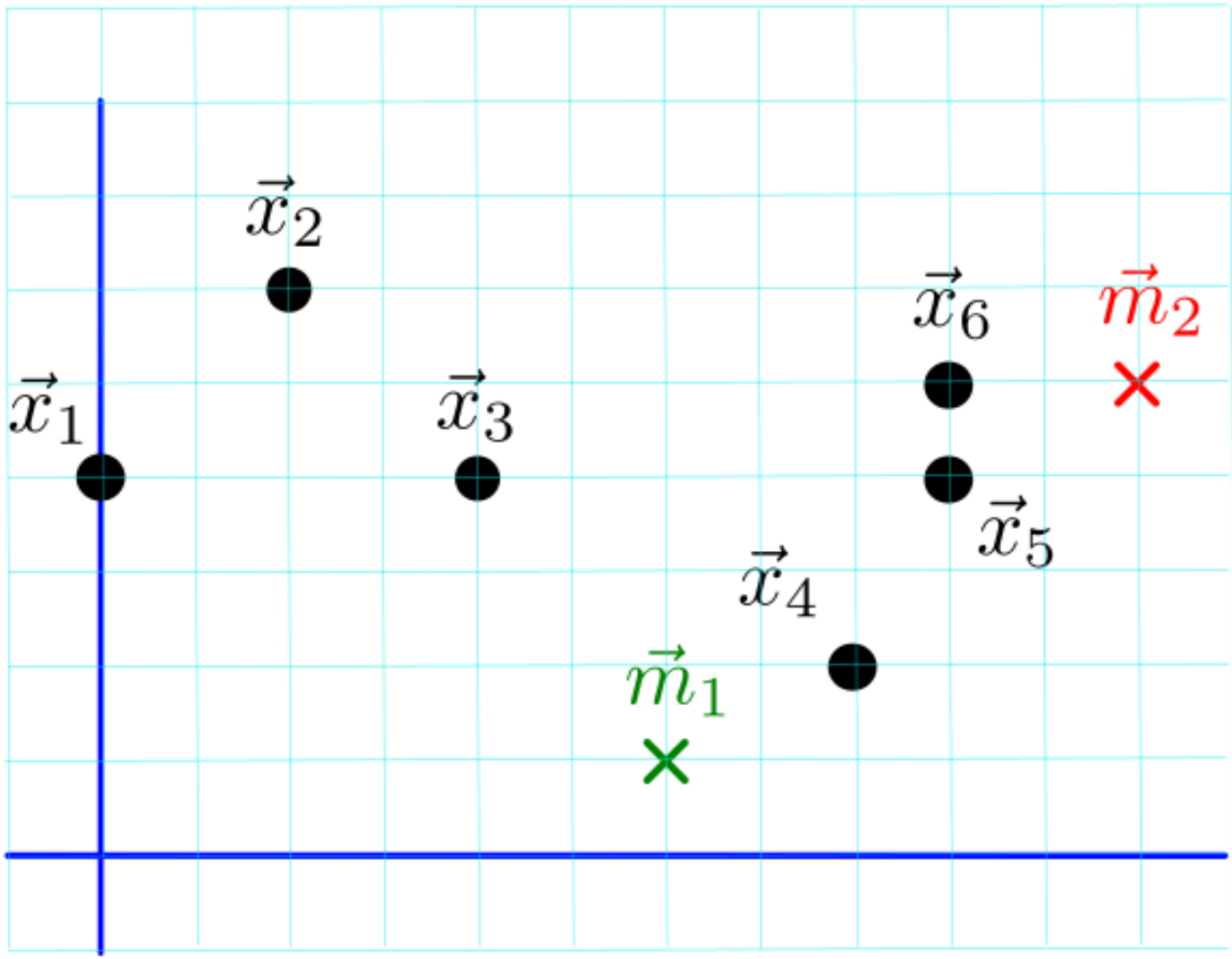
$$j = \arg\min_j d(\vec{x}_i, \vec{m}_j)^2$$

5: **end for**
6: **for** each cluster prototype $\vec{m}_j$ **do**
7: Update $\vec{m}_j$ to be the centroid of all points assigned to it

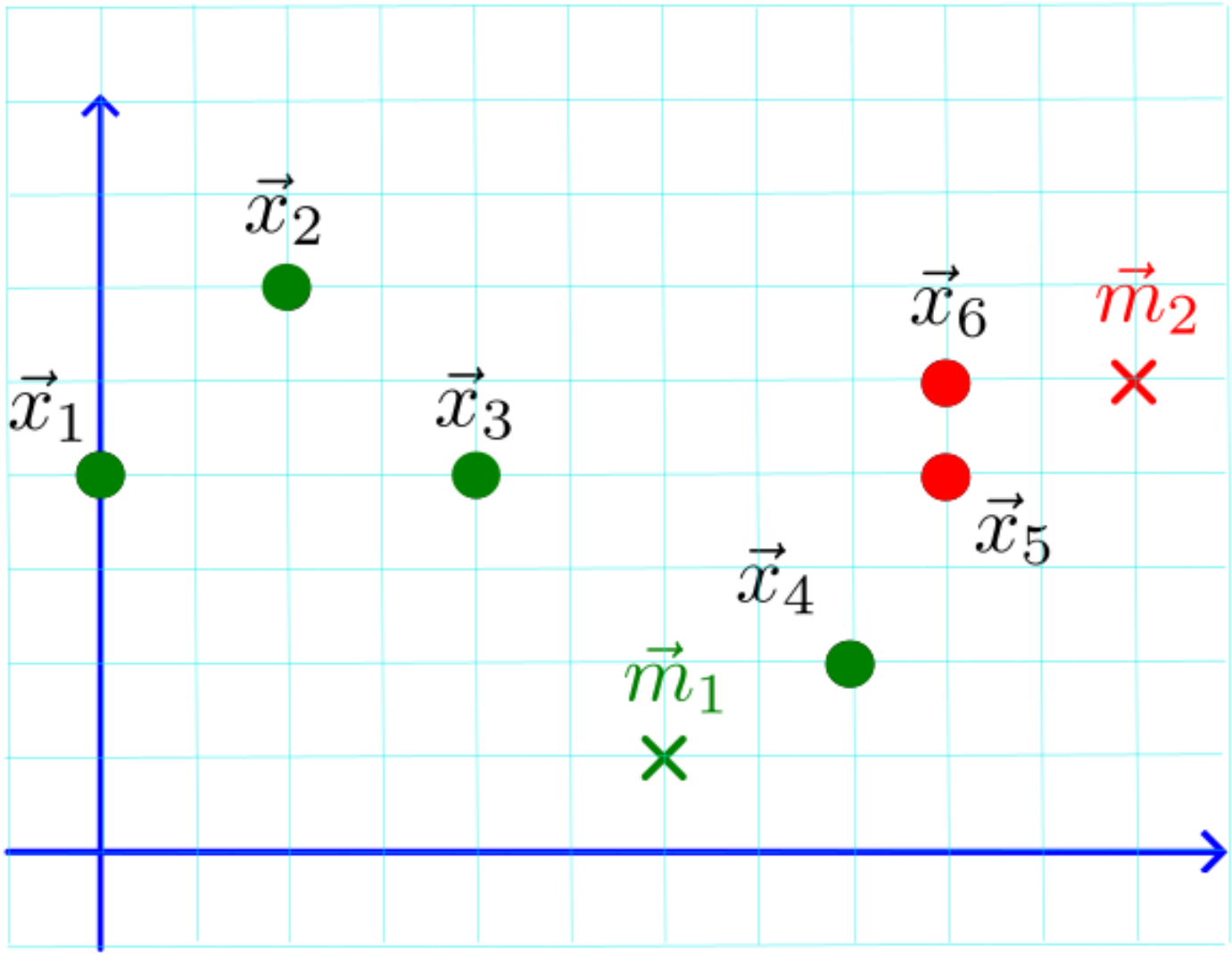
$$\vec{m}_j = \frac{1}{\sum_{i=1}^{p} q_{ij}} \sum_{i=1}^{p} q_{ij} \vec{x}_i$$

8: **end for**
9: **until** convergence

# Example

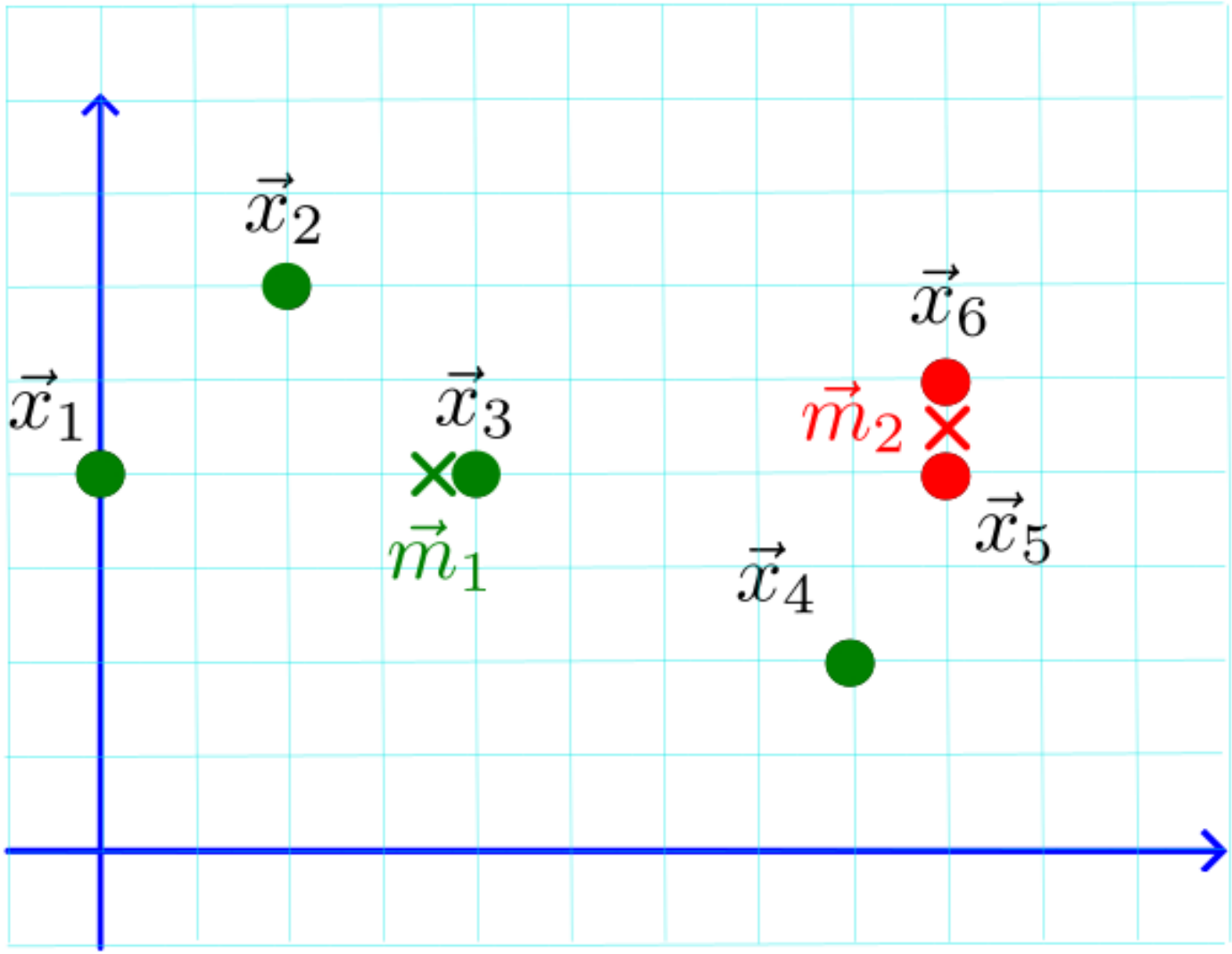


# Example



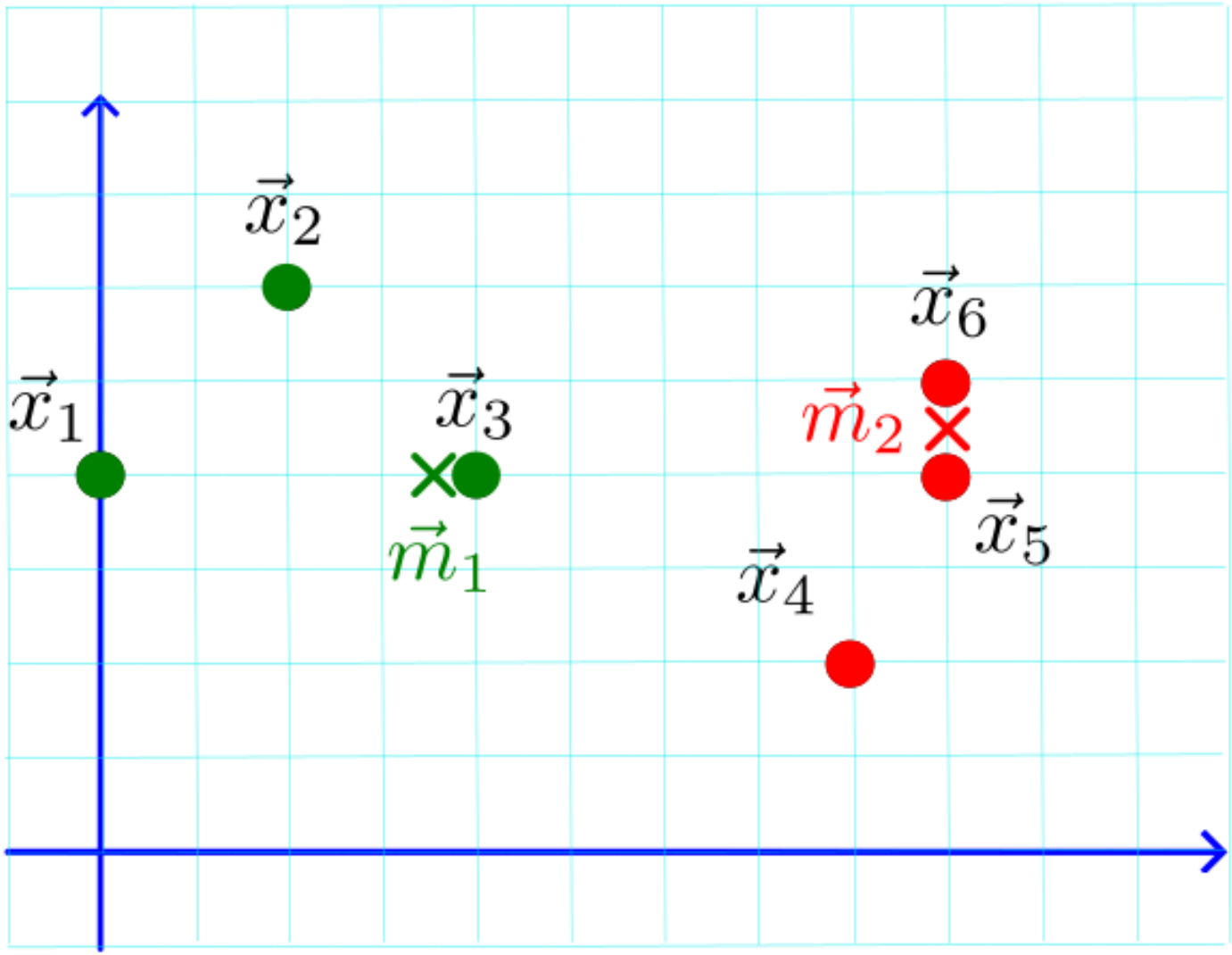

Lines 3-5: Assign examples to the prototypes.

# Example



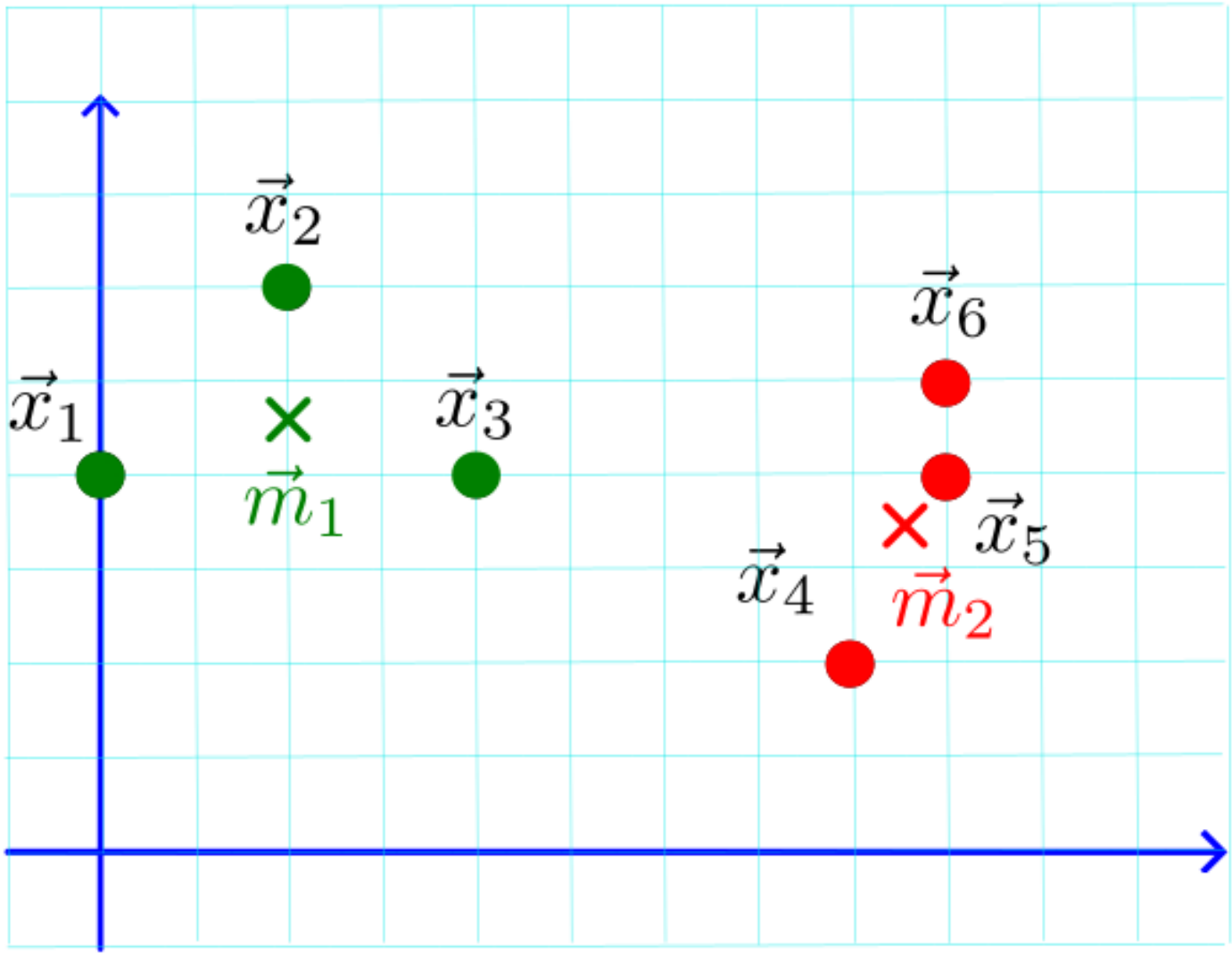

Lines 6-8: Move the prototypes to the centroids of their examples.

# Example

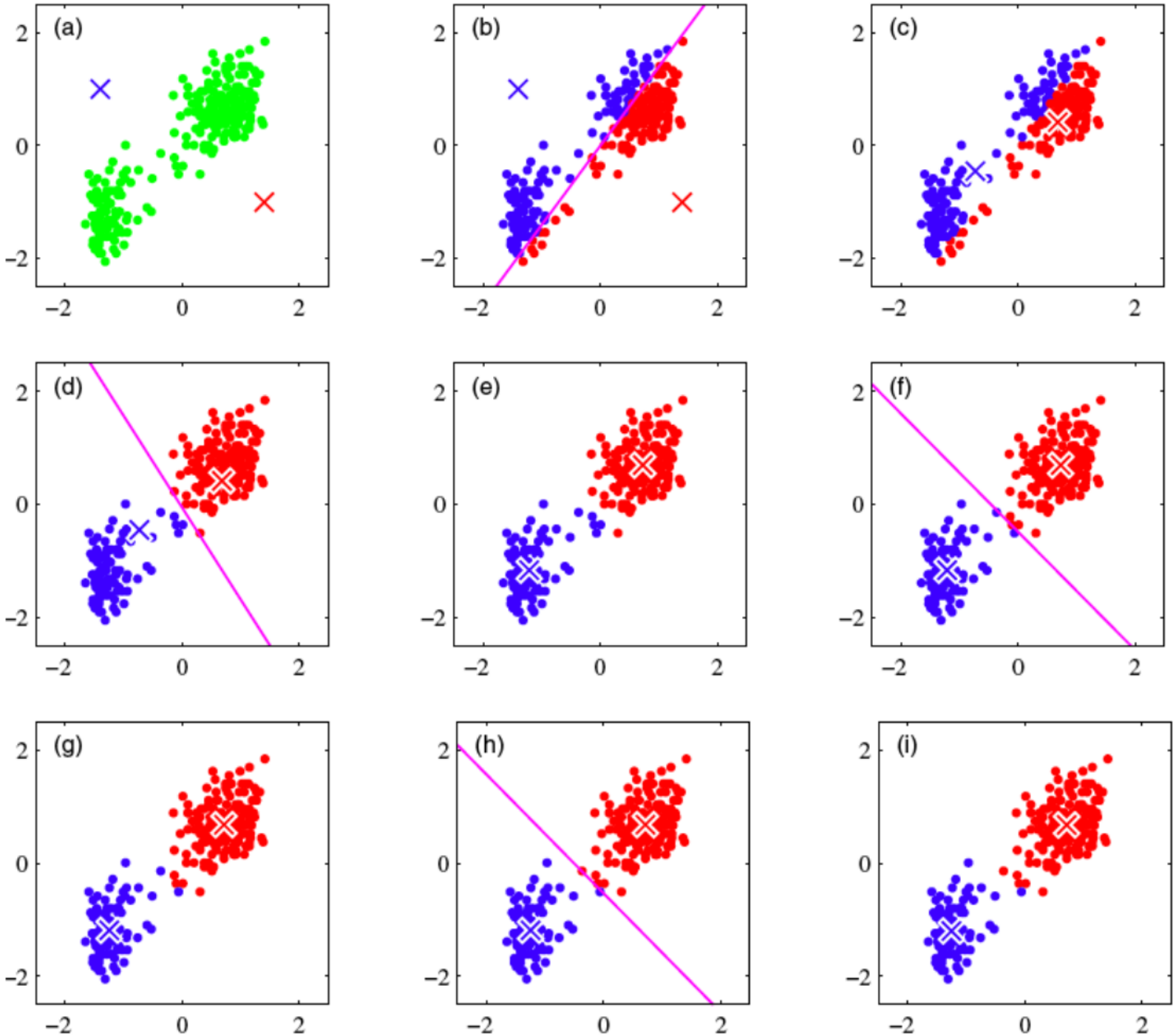



Lines 3-5: Assign examples to the prototypes.

# Example




Lines 6-8: Move the prototypes to the centroids of their examples.


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
2
0
−2
−2
0
2

# Convergence of K-means Clustering

Every step of K-means reduces the error $E(\{q_{ij}\}, \{\vec{m}_j\})$:

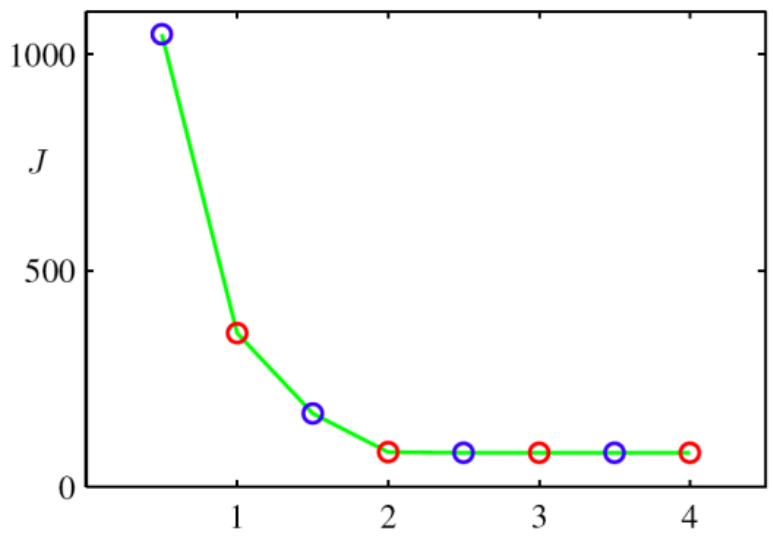
- We always assign an input vector to the closest prototype.
- We always move the prototype to be "closest" to the input vectors it represents.

Convergence can be tested by computing the error and checking whether it has not changed in the last step.

This will always happen after finitely many steps.

There are only finitely many possible assignments to $q_{ij}$, and we always minimize the distance of inputs to their assigned centers.

Example error development during training. Blue circles mean reassignment, and red circles mean moving prototypes.



# Setting K - the Elbow Method

$K$-means clustering minimizes the *inertia* measure:

$$E(\{q_{ij}\}, \{\vec{m}_j\}) = \sum_{i=1}^{p} \sum_{j=1}^{k} q_{ij} d(\vec{x}_i, \vec{m}_j)^2$$

That is the sum of squared distances of all examples of $D$ to the cluster prototypes.

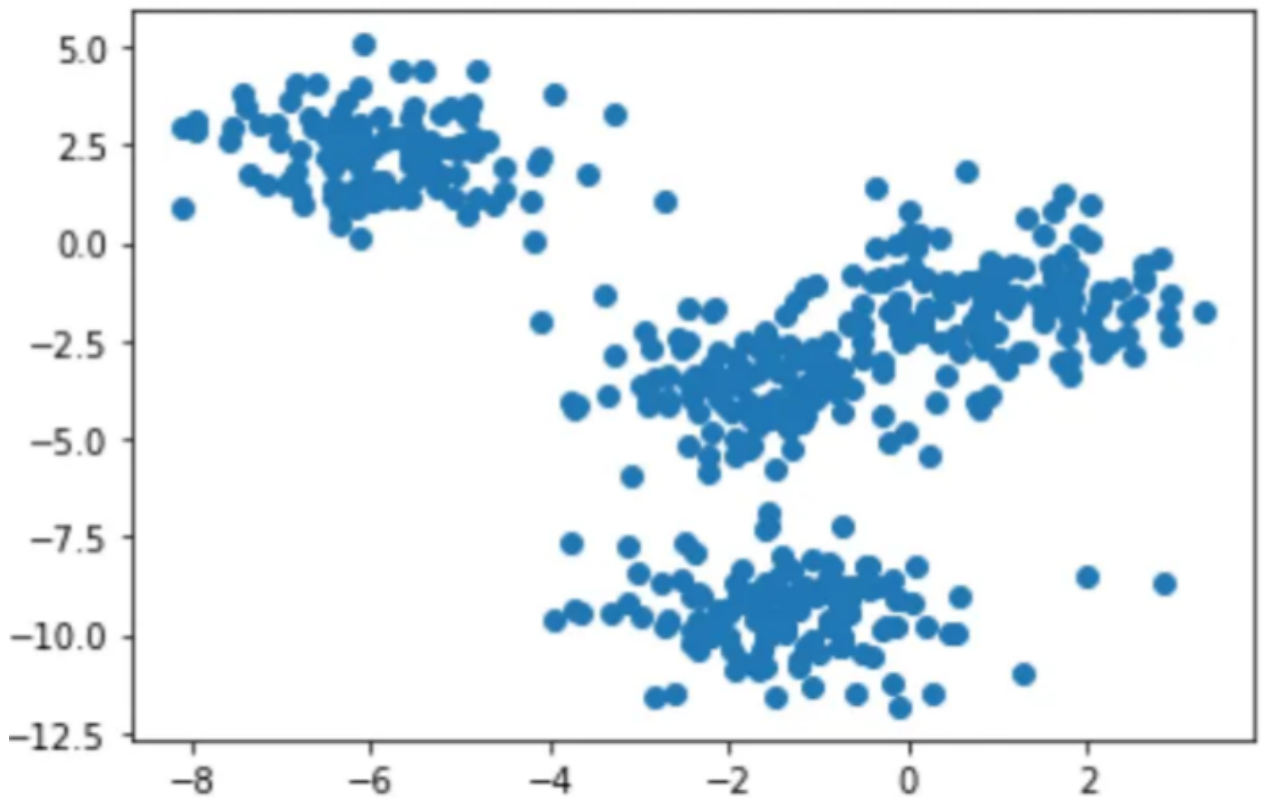
Note that the error does not consider the distance between the centers of the clusters.

Still, it is a valid measure that can be used to select the number of clusters.

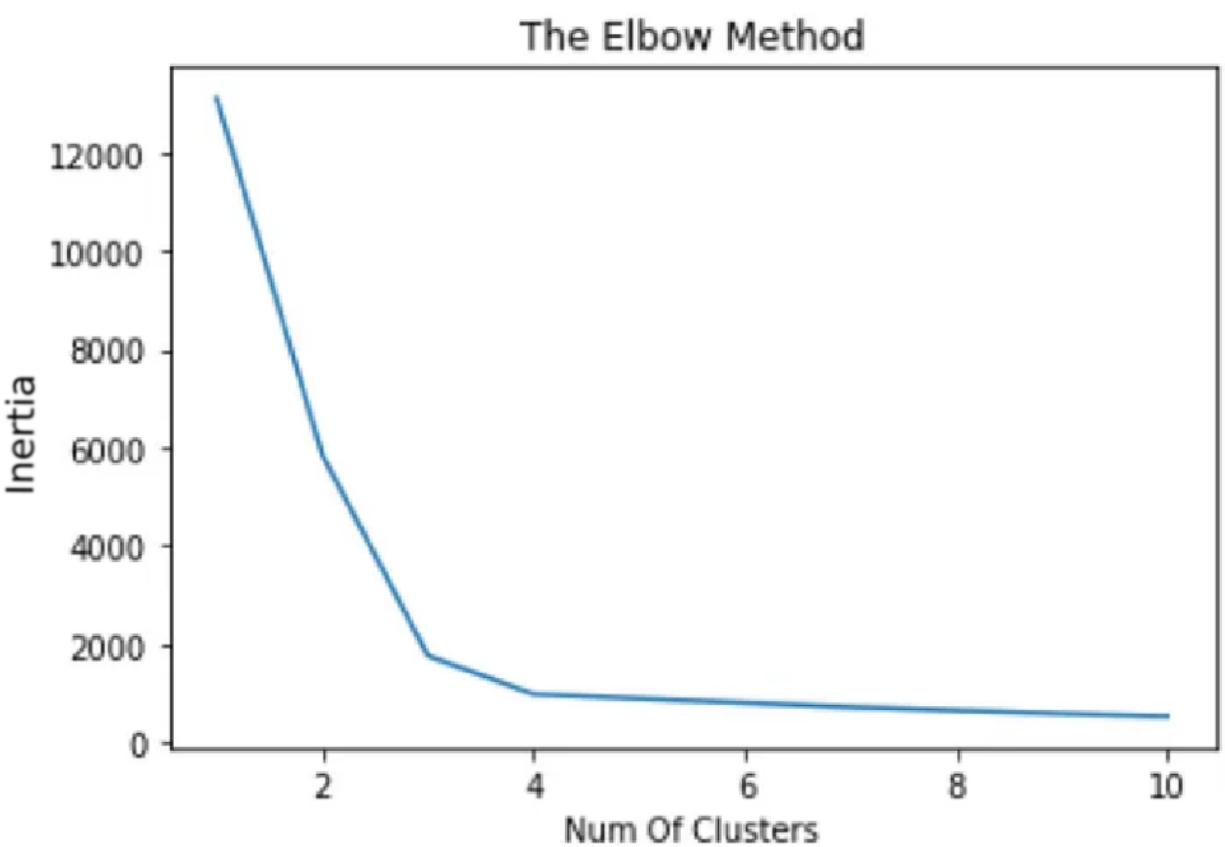
# Elbow Method

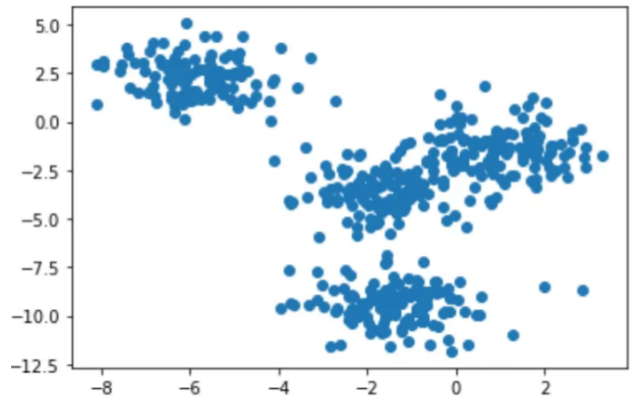
The following method for setting up the hyperparameters can be used in general. Let us illustrate the elbow method on $K$-means clustering with the inertia measure.

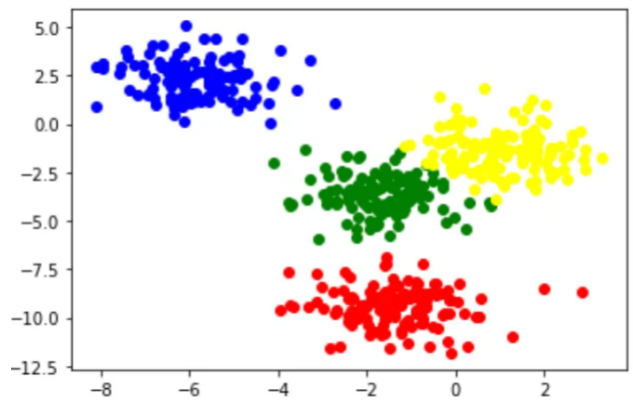
Consider the following data:



# Elbow Method



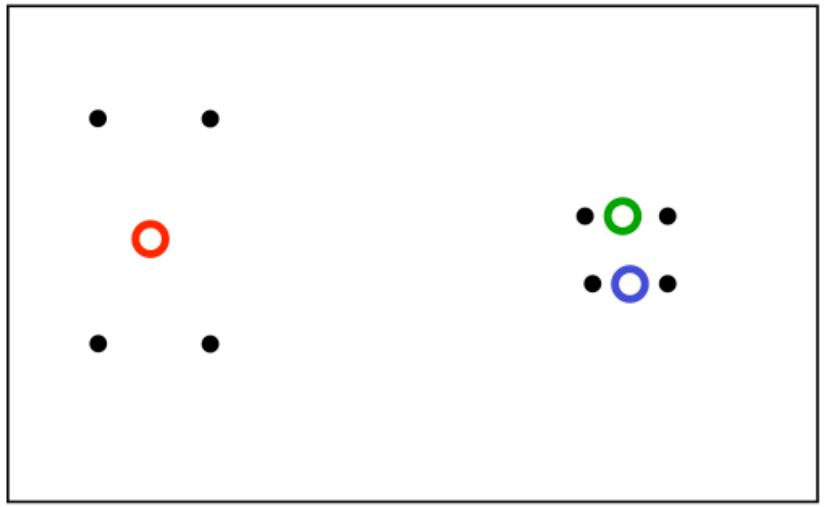




We could choose four clusters because adding more leads only to small decrements in the inertia.

# Bad Behavior

Minimizing $E(\{q_{ij}\}, \{\vec{m}_j\})$ starting from random positions of prototypes does not always produce "nice" results.

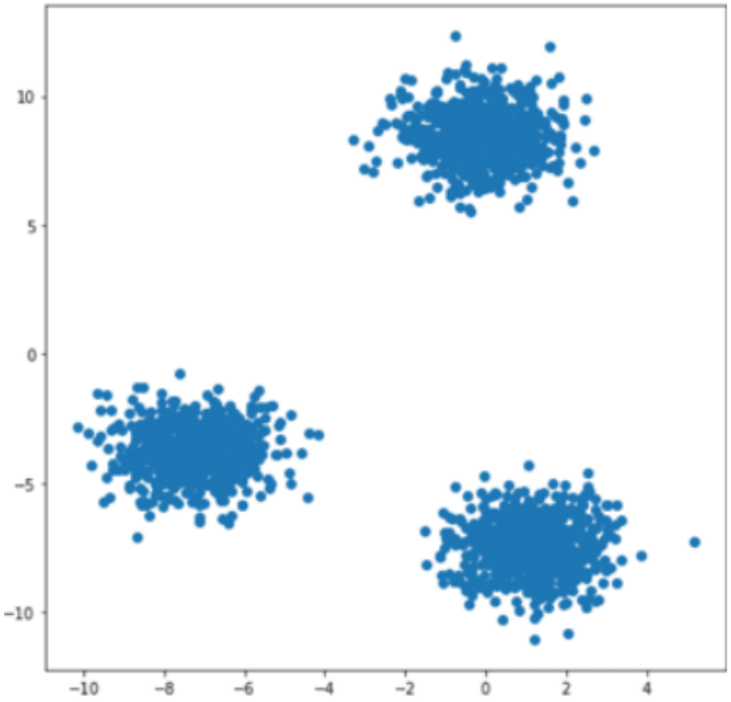
Some runs correspond to apparently bad solutions to the clustering problem even though a better solution exists.

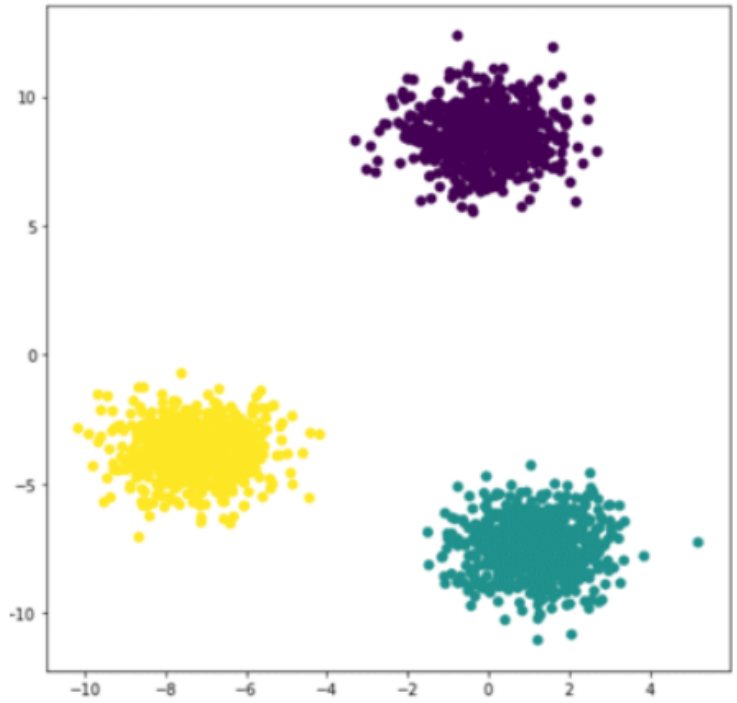


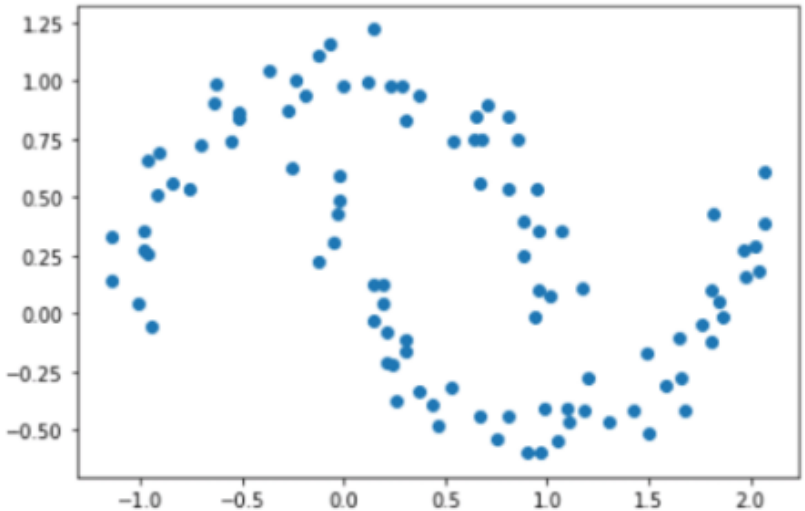
Possible solution: Start the algorithm several times with random initialization of the prototypes.

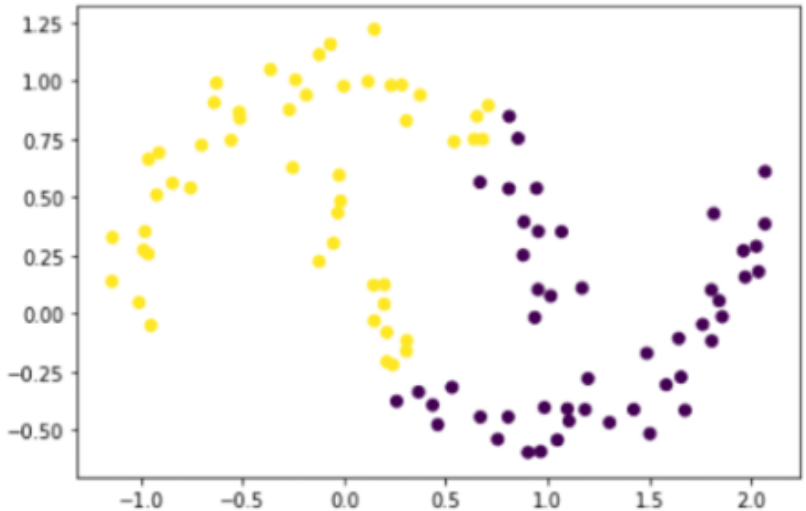
# Properties of K-means Clustering

- Prototype initialization is a big issue in K-means. There are various strategies. For example:
  - Start with all centers in a single corner.
  - Include randomness in the setting of centers throughout the algorithm.
  - Initialize sequentially, always fit prototypes, and then choose a new one as far away from the others as possible.
  - Use hierarchical clustering (next slides) to find clusters and initialize $K$-means with their centroids.
- Empty clusters may occur - need to resolve, e.g., by assigning the farthest point from any current prototype.
- As the squared error is behind the basic method, outliers may strongly affect its behavior (as in the linear regression case).
- Other problematic properties of data include
  - non-convex clusters
  - clusters of different sizes
  - non-linearly separable clusters
  - overlapping clusters


10
5
0
−5
−10
−10
−8
−6
−4
−2
0
2
4


10
5
0
−5
−10
−10
−8
−6
−4
−2
0
2
4


1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

# Agglomerative Clustering

# Agglomerative Clustering

Consider a dataset

$$D = \{\vec{x}_1, \ldots, \vec{x}_p\}$$

Here $\vec{x}_i \in \mathbb{R}^n$ for all $i = 1, \ldots, p$. Assume a distance $d$ (e.g., Euclidean).

**Idea:**

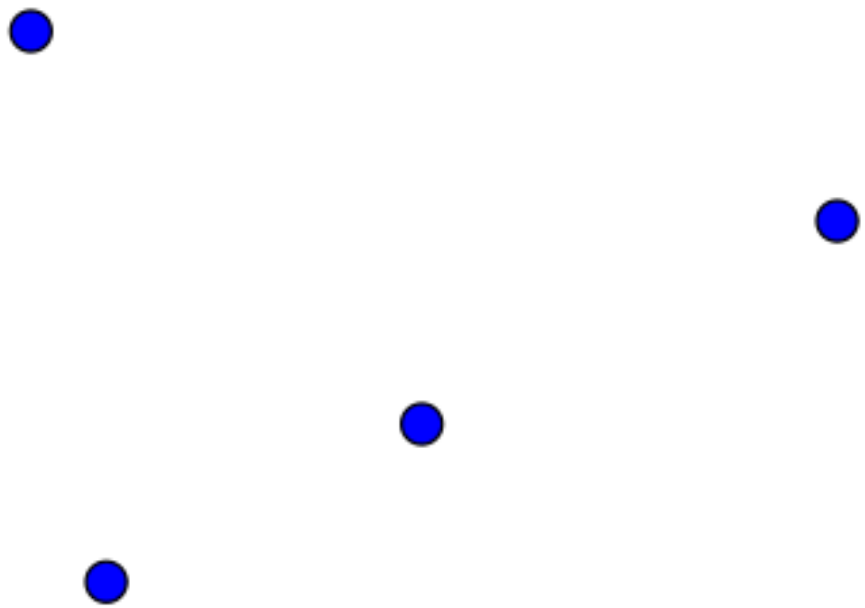
- Start by merging the closest examples (w.r.t. $d$)
- Incrementally build *larger clusters* by merging smaller clusters.

**More concretely:**

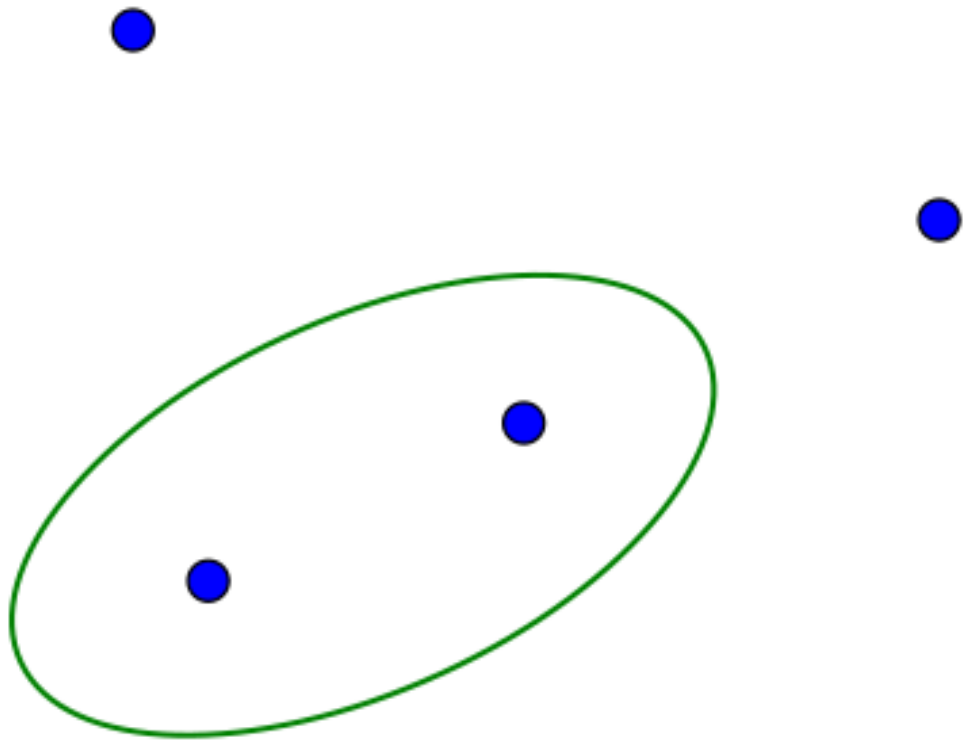
- Maintain a set of clusters
- Initially, each $\vec{x}_i$ in its own cluster
- Repeat until only one cluster is left:
  - Pick two *closest clusters*
  - Merge them into a new cluster

How do we determine the closest clusters?

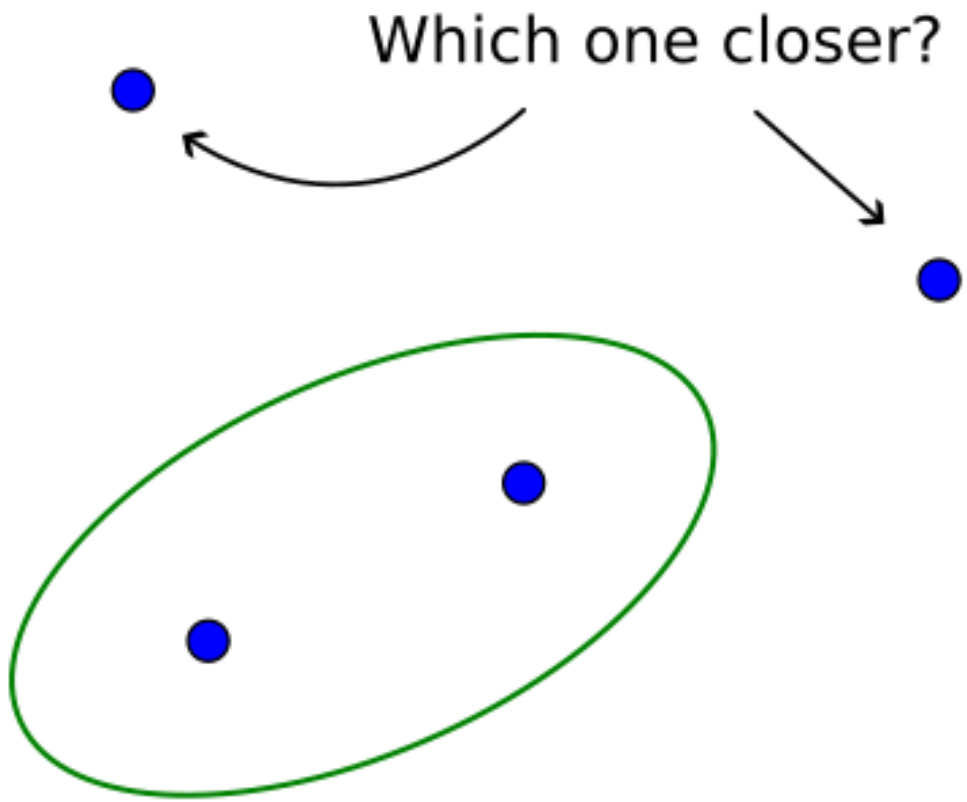
# Closest Clusters



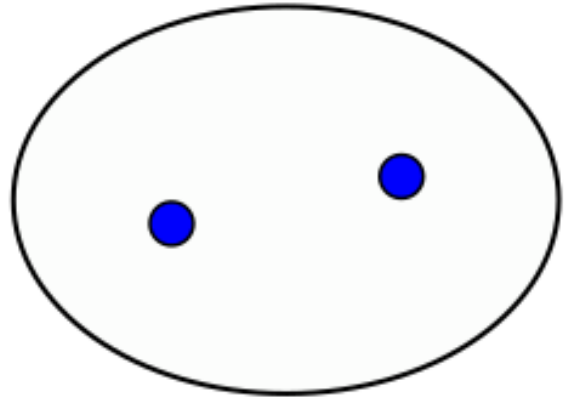
# Closest Clusters

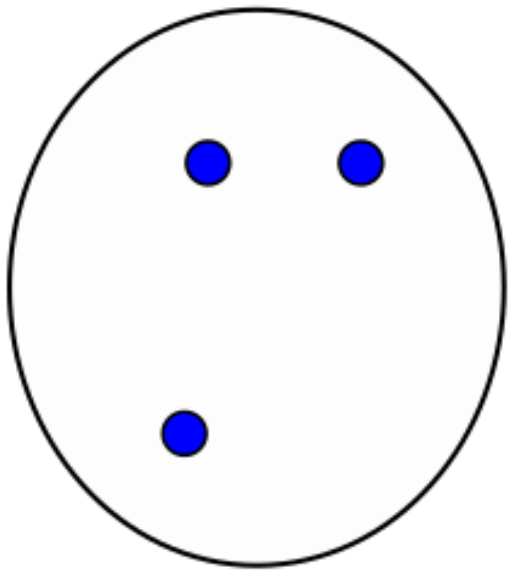


# Closest Clusters

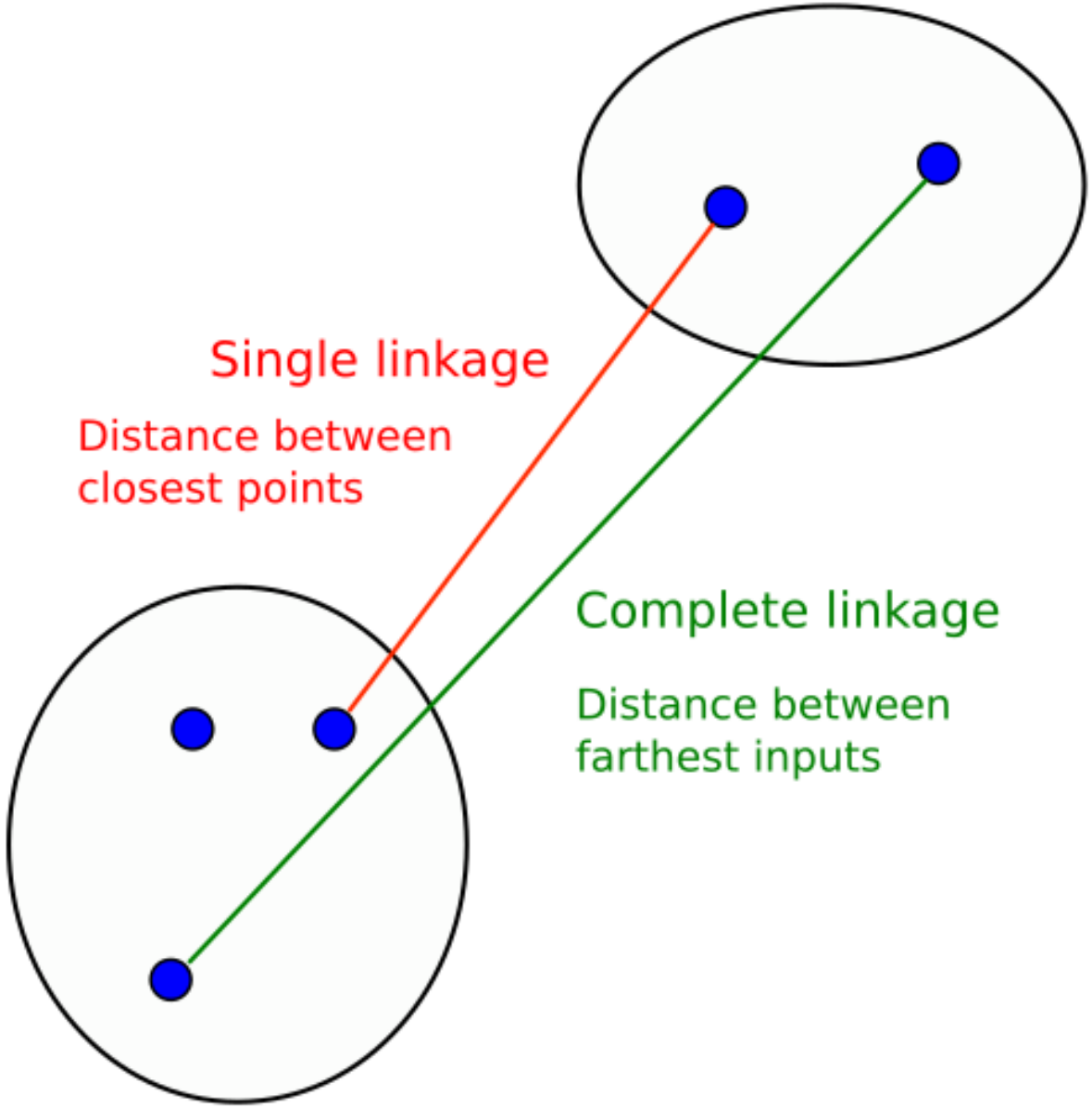


# Distance Between Clusters

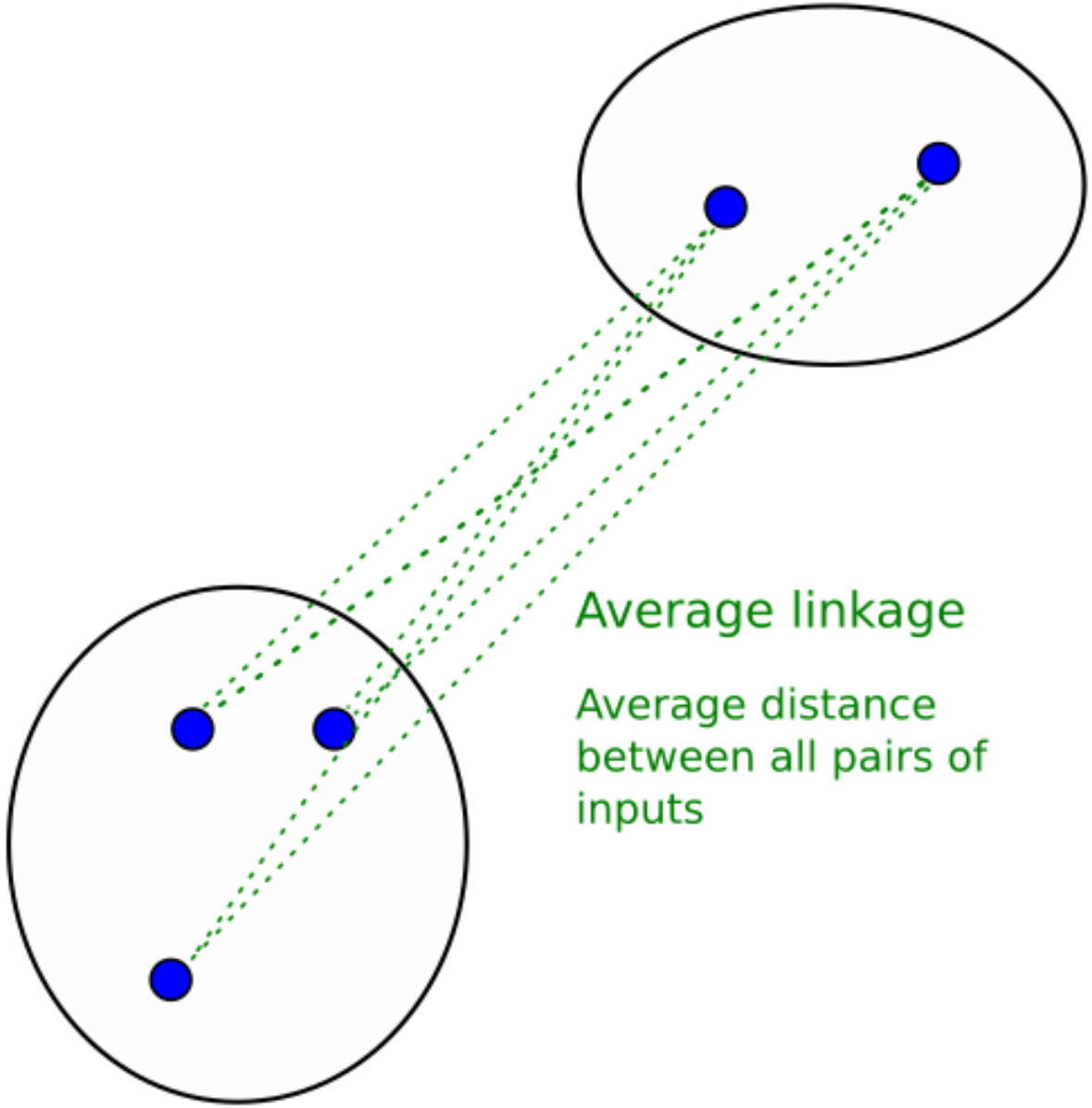




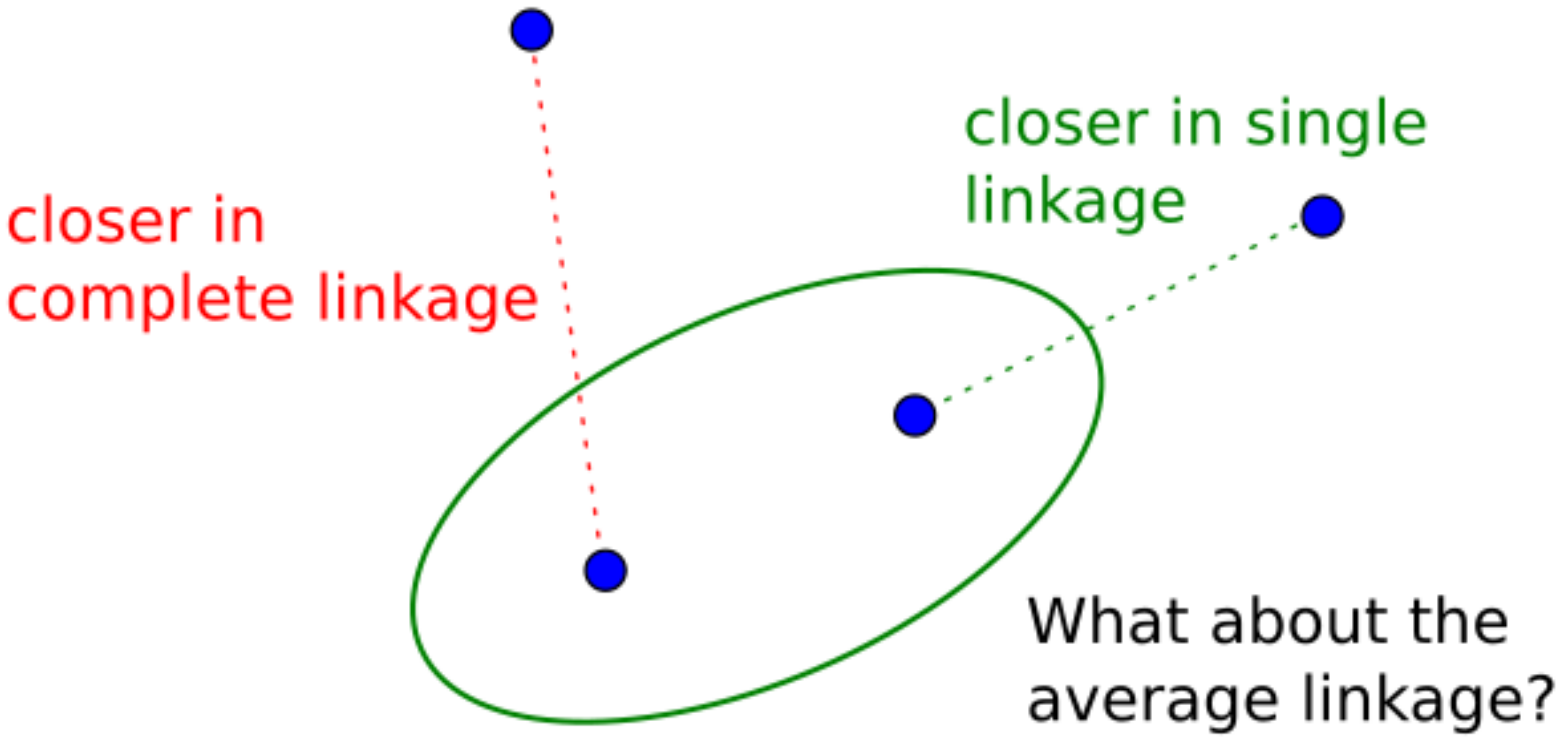
# Distance Between Clusters



# Distance Between Clusters



# Which One is Closer?



# Distance Between Clusters

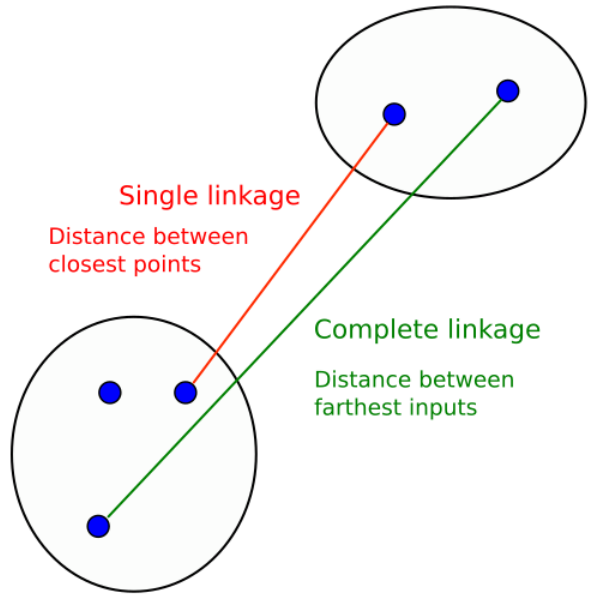
Consider two clusters $U_j, U_k \subseteq D$.

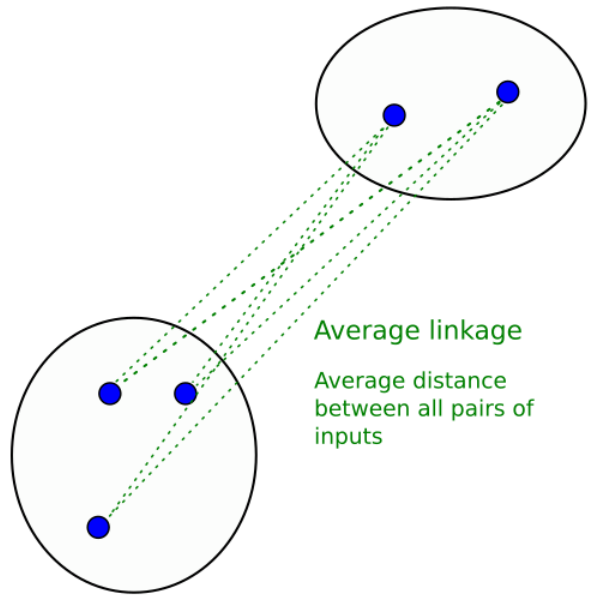
$$\text{single\_linkage}(U_j, U_k) = \min\{d(\vec{x}, \vec{z}) \mid \vec{x} \in U_j, \vec{z} \in U_k\}$$

$$\text{complete\_linkage}(U_j, U_k) = \max\{d(\vec{x}, \vec{z}) \mid \vec{x} \in U_j, \vec{z} \in U_k\}$$

$$\text{average\_linkage}(U_j, U_k) = \frac{1}{|U_j||U_k|} \sum_{\vec{x} \in U_j} \sum_{\vec{z} \in U_k} d(\vec{x}, \vec{z})$$

Each linkage can result in a different clustering.

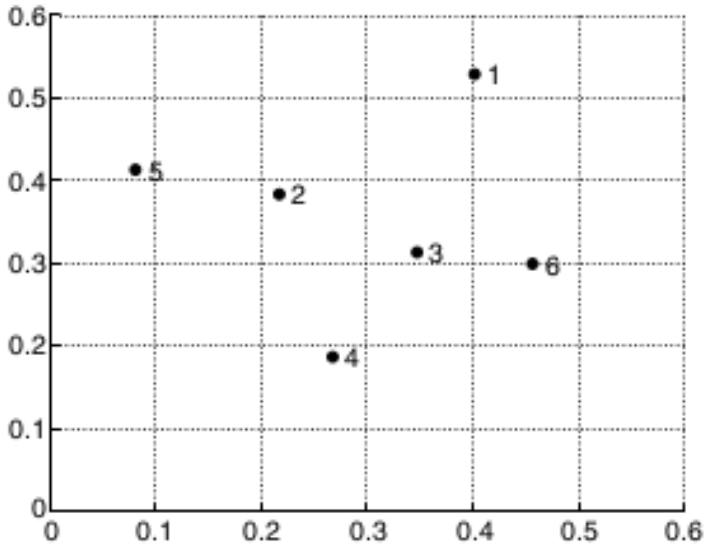


# Agglomerative Hierarchical Clustering Algorithm

Maintain a set of clusters
Initially, each $\vec{x}_i$ in its own cluster
**repeat**
    Pick two closest clusters
        Using the distance measure $d$ and single, average, or complete linkage.
    Merge them into a new cluster
**until** only one cluster is left

# Example

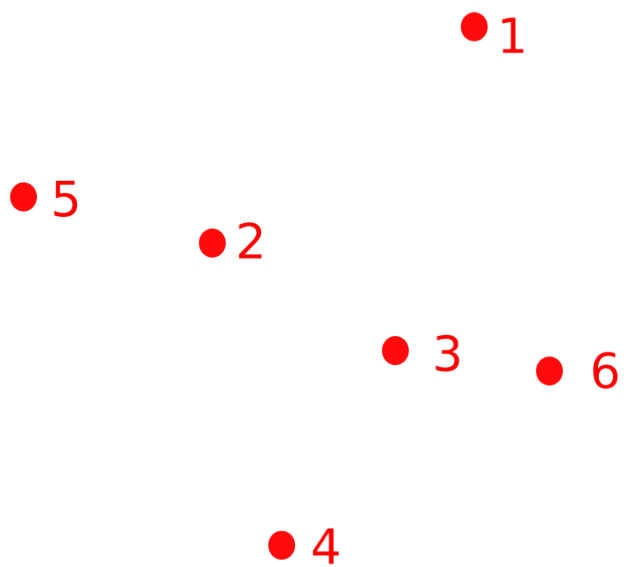



| Point | $x$ Coordinate | $y$ Coordinate |
|---|---|---|
| p1 | 0.40 | 0.53 |
| p2 | 0.22 | 0.38 |
| p3 | 0.35 | 0.32 |
| p4 | 0.26 | 0.19 |
| p5 | 0.08 | 0.41 |
| p6 | 0.45 | 0.30 |

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |

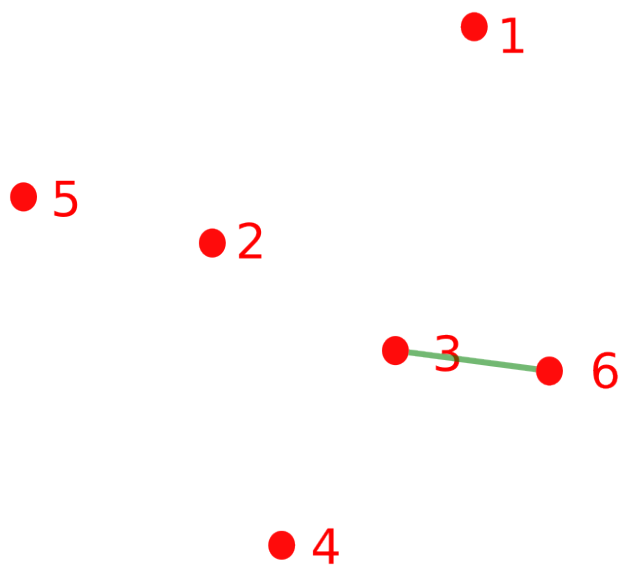
# Example - Single Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



# Example - Single Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |

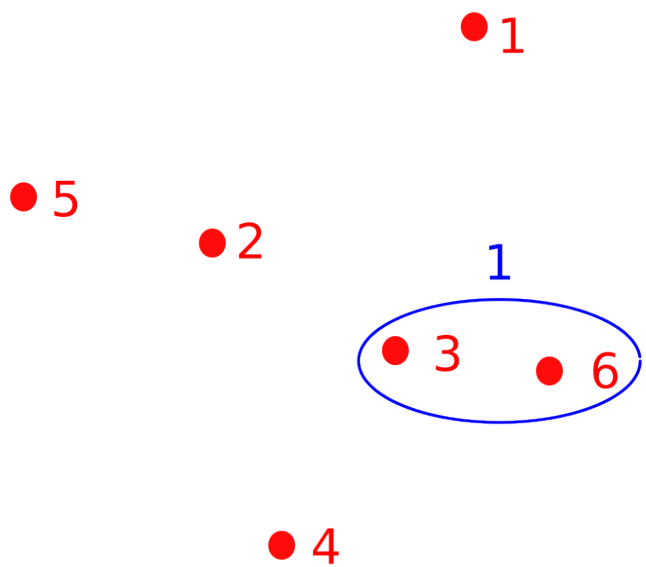



$$d(3,6) = 0.11$$

which is the minimum distance between points.

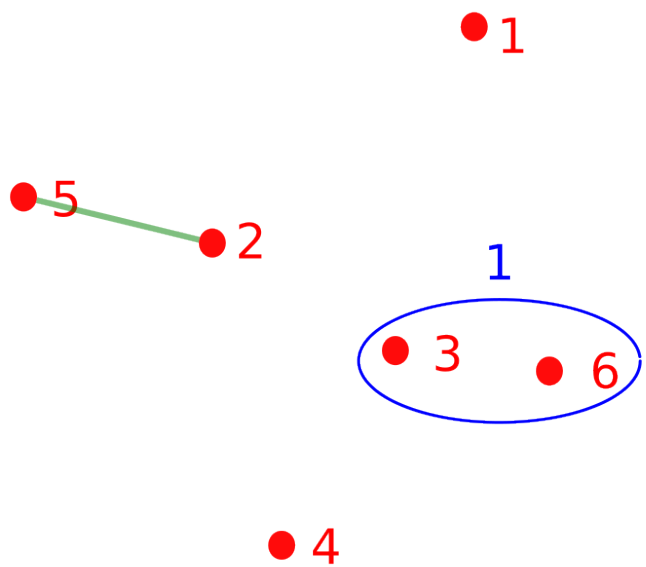
# Example - Single Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



# Example - Single Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |

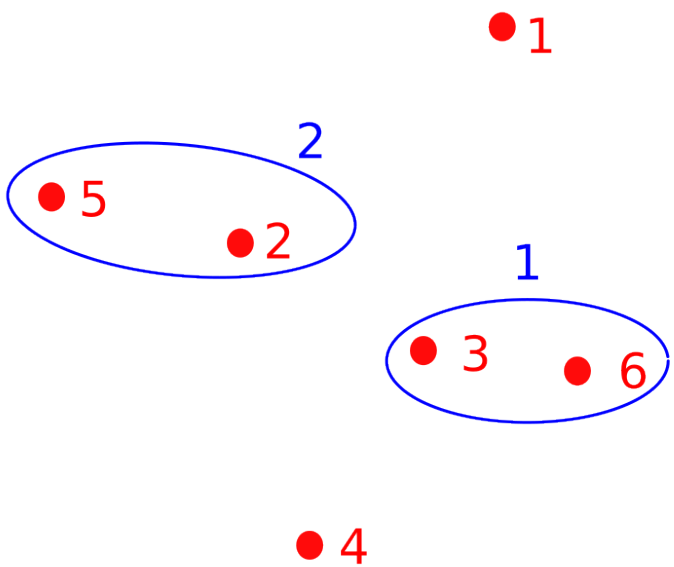



$$d(2,5) = 0.14$$

which is the second smallest distance.

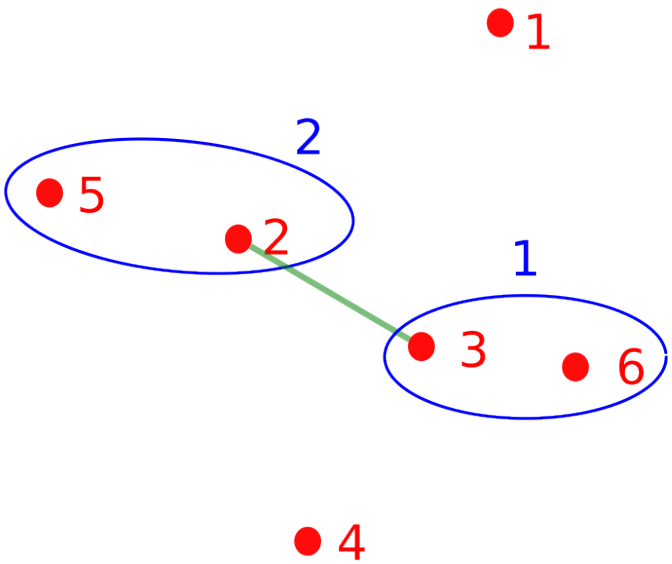
# Example - Single Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



# Example - Single Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |

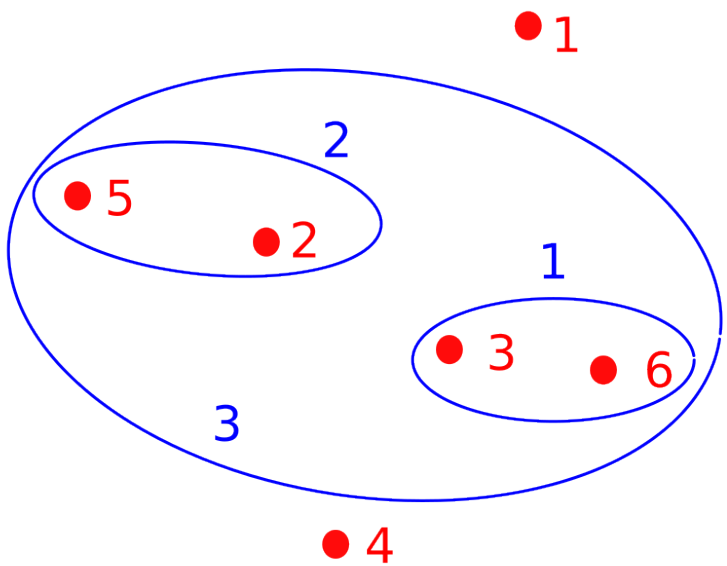



$$d(2,3) = 0.15 =$$
$$\min\{d(2,3), d(2,6), d(5,3), d(5,6)\}$$

which is smaller than
$d(1,2) = 0.24$,
$d(1,3) = 0.22$,
$d(4,2) = 0.2$,
$d(4,3) = 0.16$,
$d(4,1) = 0.37$
the min. distances of points
in all other pairs of clusters.

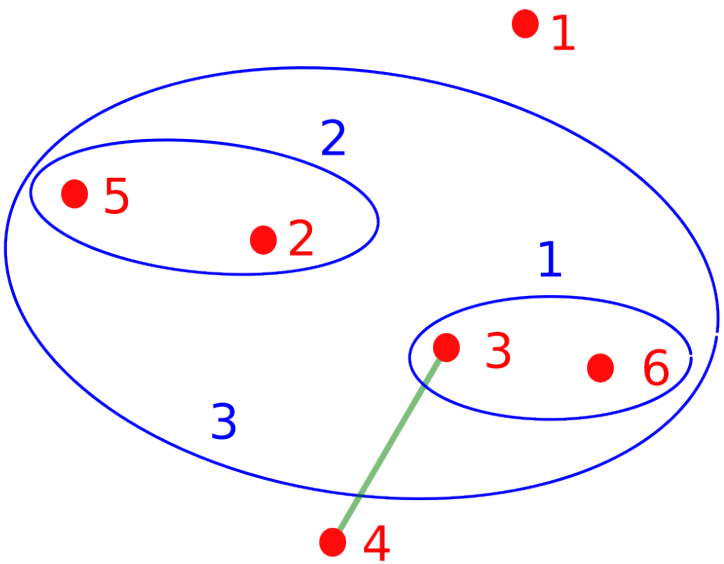
# Example - Single Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



# Example - Single Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |

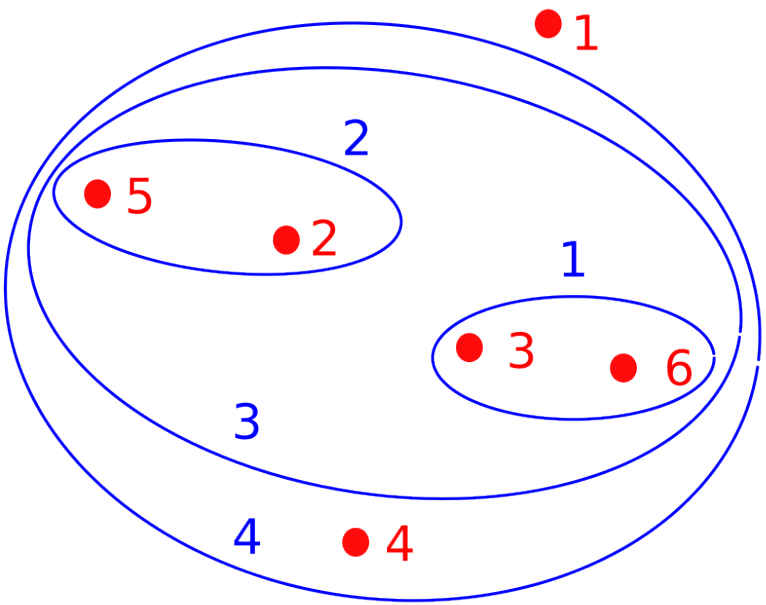



$$d(4,3) = 0.15$$
$$= \min\{d(4,3), d(4,5), d(4,2), d(4,6)\}$$

which is smaller than
$d(1,3) = 0.22$, the distance
of 1 to the cluster 3.

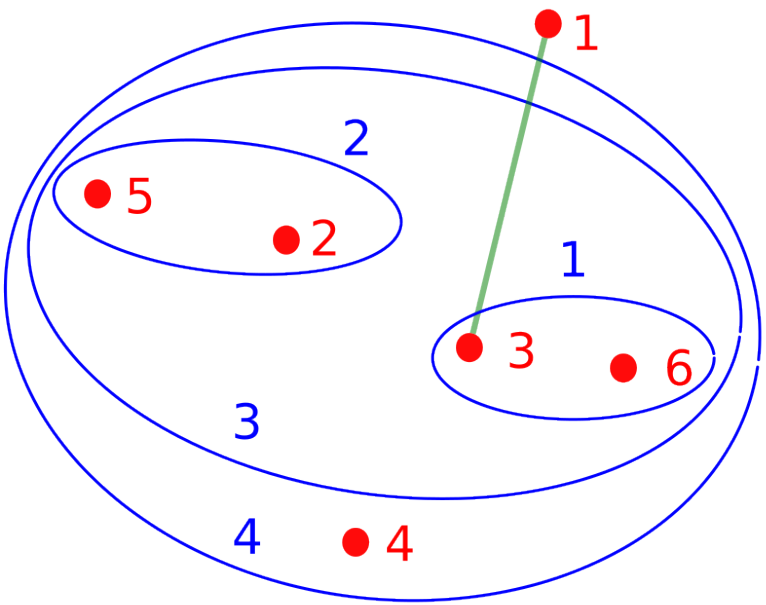
# Example - Single Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



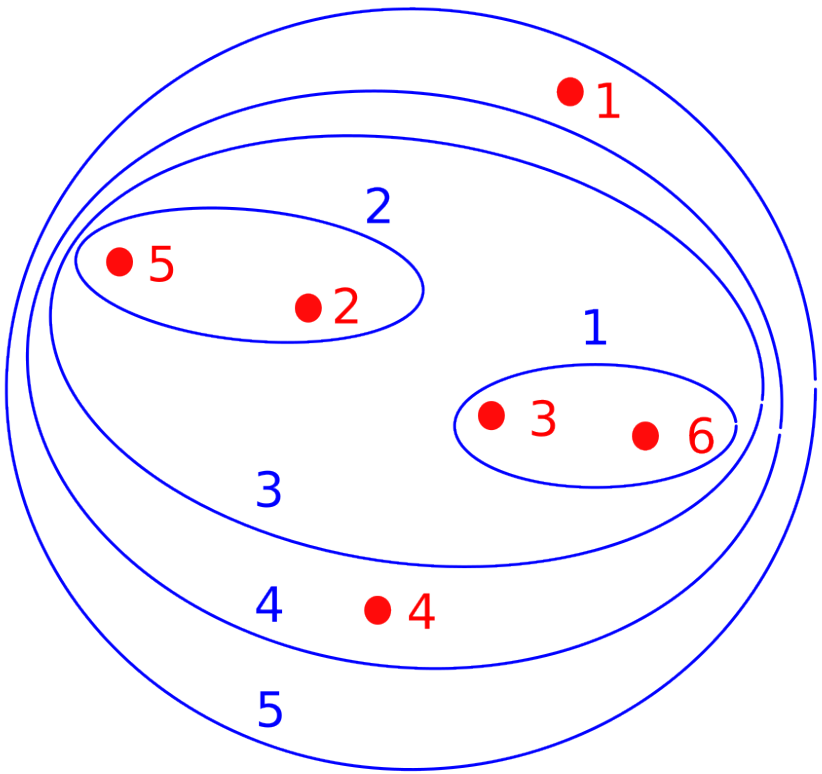
# Example - Single Linkage

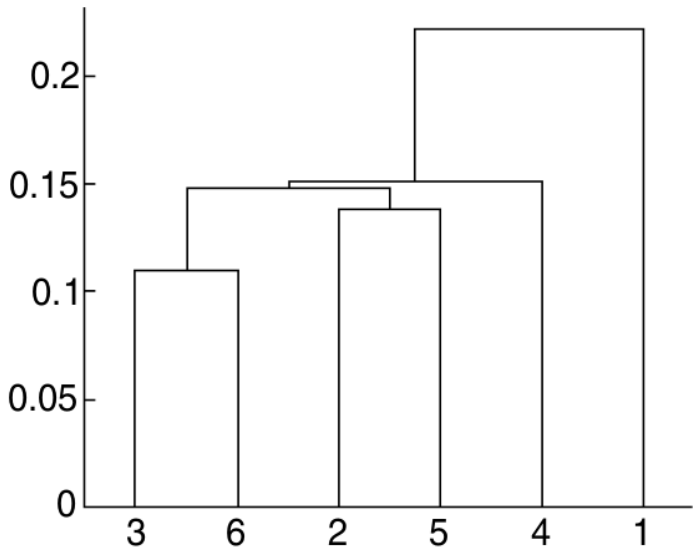
| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



# Example - Single Linkage

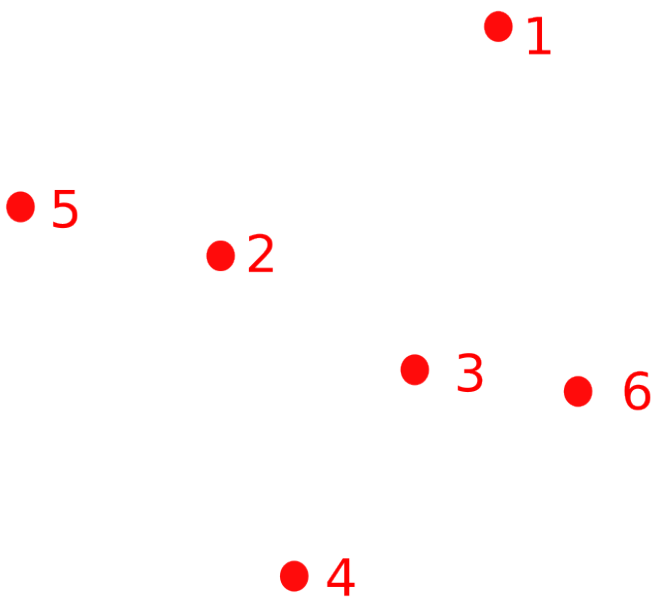
| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



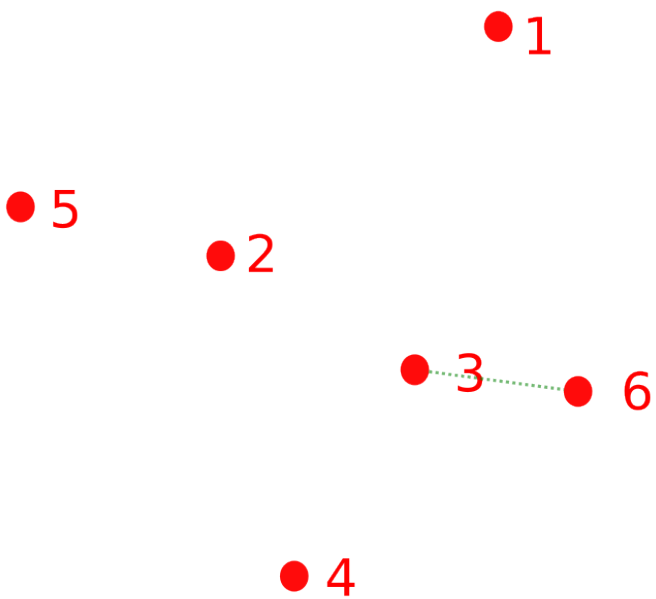
# Example - Average Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



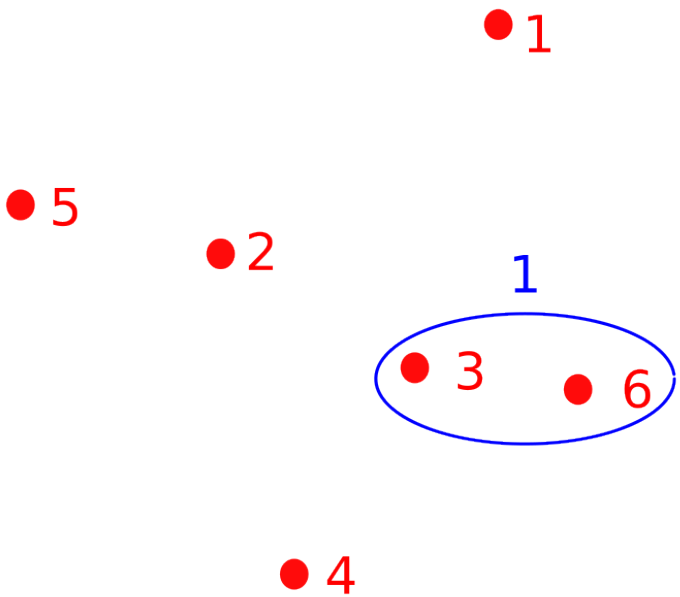
# Example - Average Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



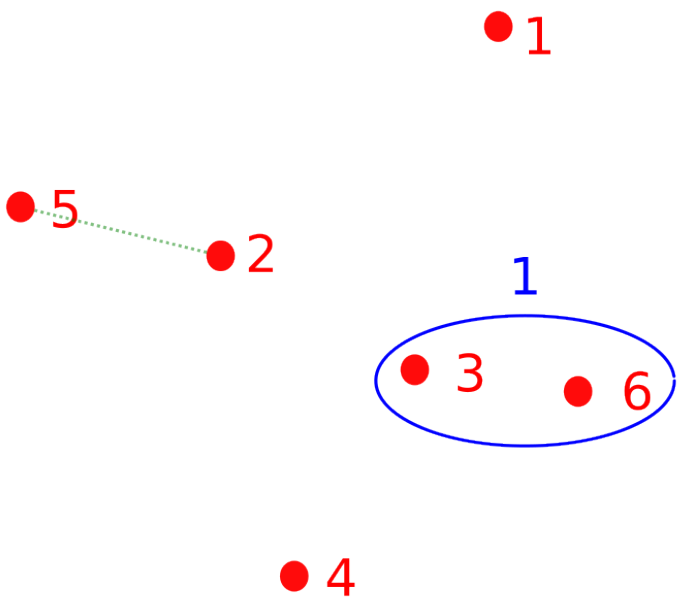
# Example - Average Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



# Example - Average Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |

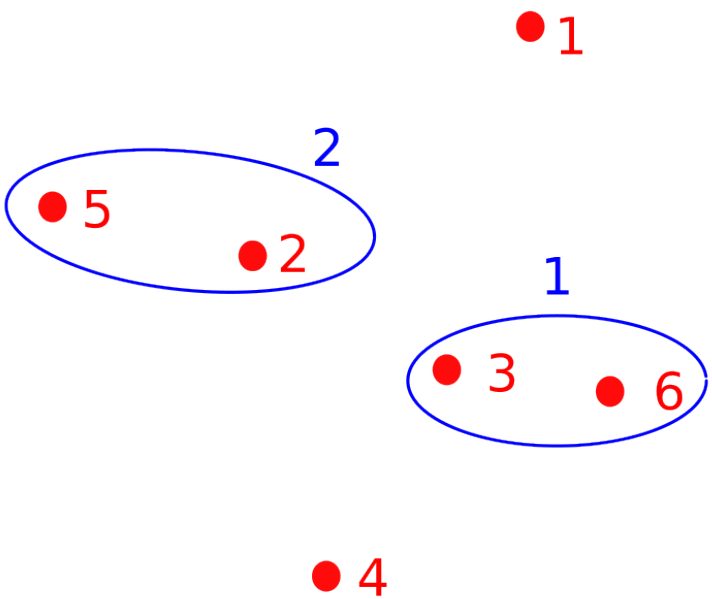



$$d(2,5) = 0.14$$

which is second smallest distance.

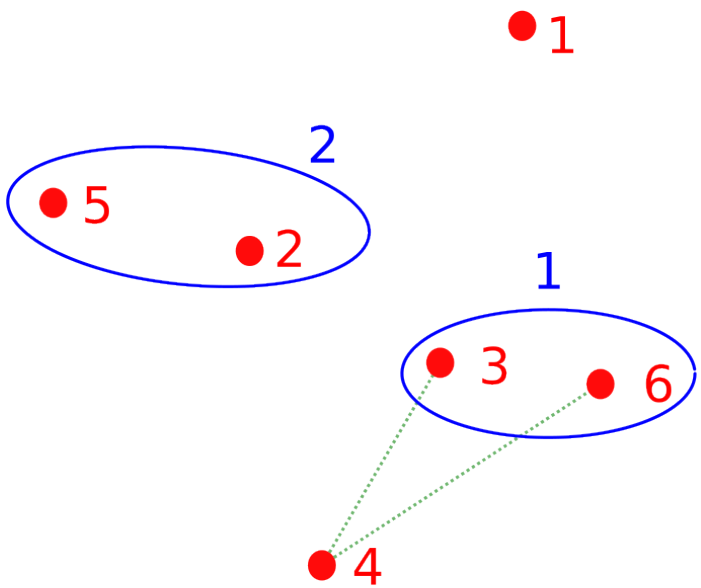
# Example - Average Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



# Example - Average Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |




The average distance between 4 and both points of $\{3, 6\}$ is

$$\frac{1}{2}(d(4,3) + d(4,6)) = 0.19$$

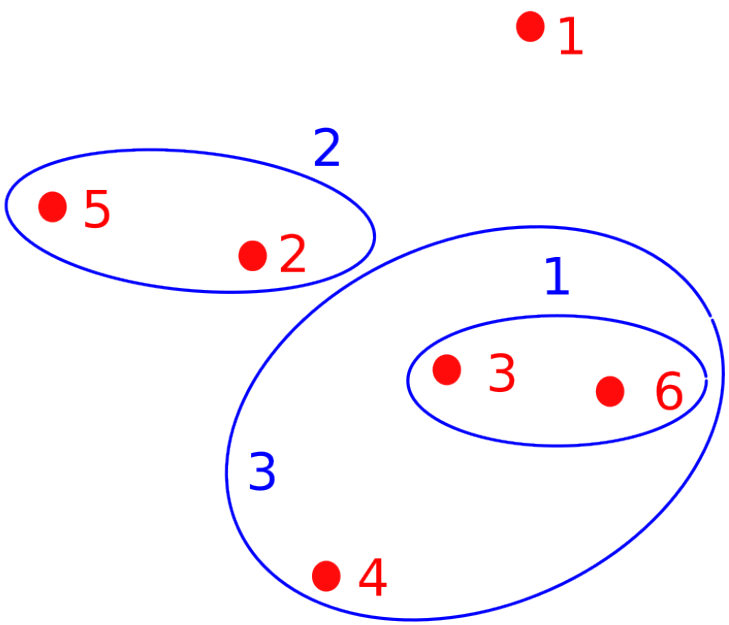
which is smaller than the average distance between all points of clusters $1, 2$:

$$\frac{d(5,2) + d(5,3) + d(2,3) + d(2,6)}{4}$$

(equal to 0.205), and the average distance of 1 to any cluster.

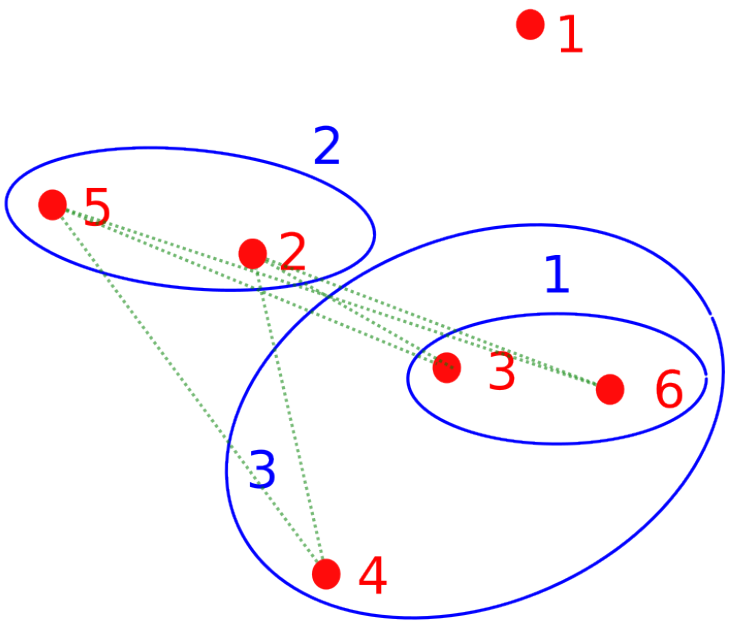
# Example - Average Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



# Example - Average Linkage

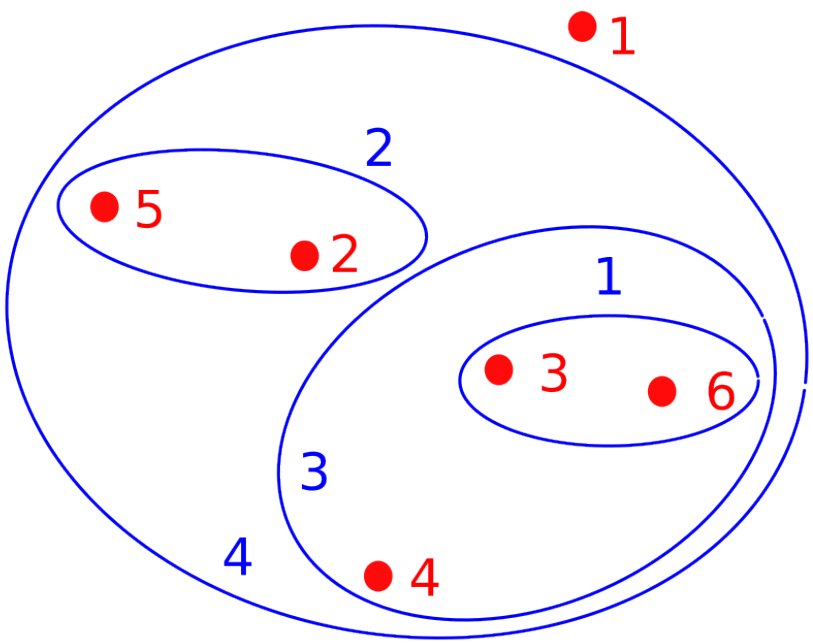
| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |




The average distance between clusters $2, 3$ is 0.26 which is smaller than the average distance of $1$ to any of the two clusters $1, 2$ (the average distances are 0.273 and 0.29).

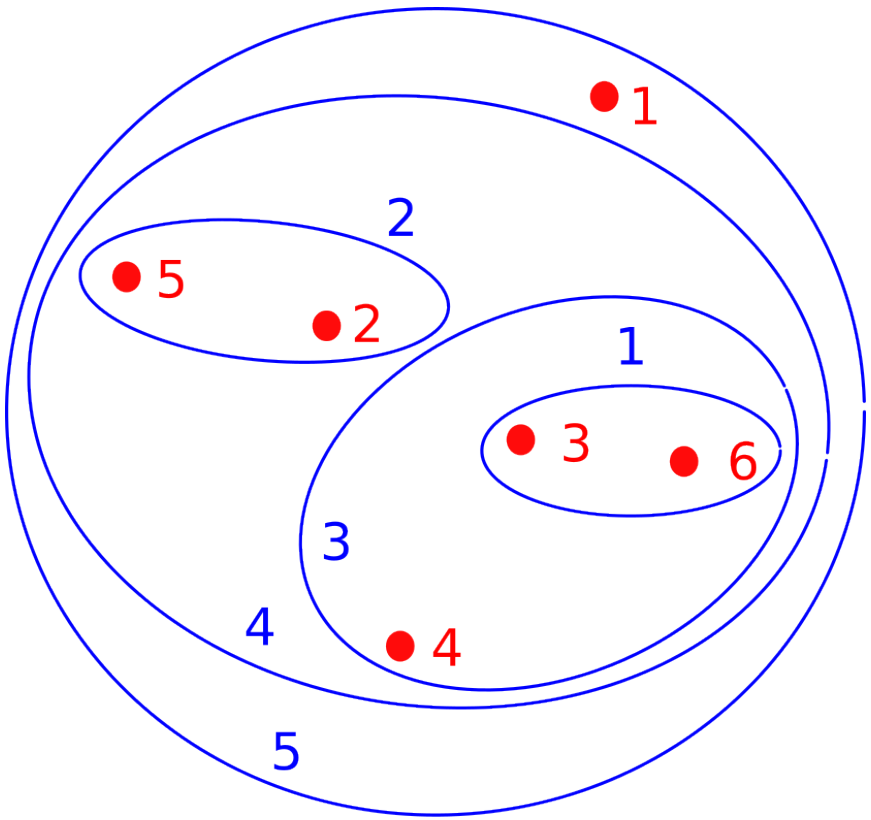
# Example - Average Linkage

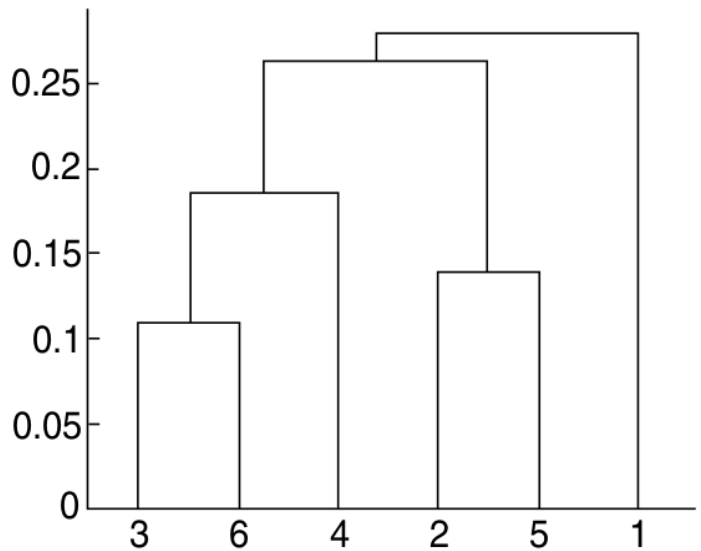
| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



# Example - Average Linkage

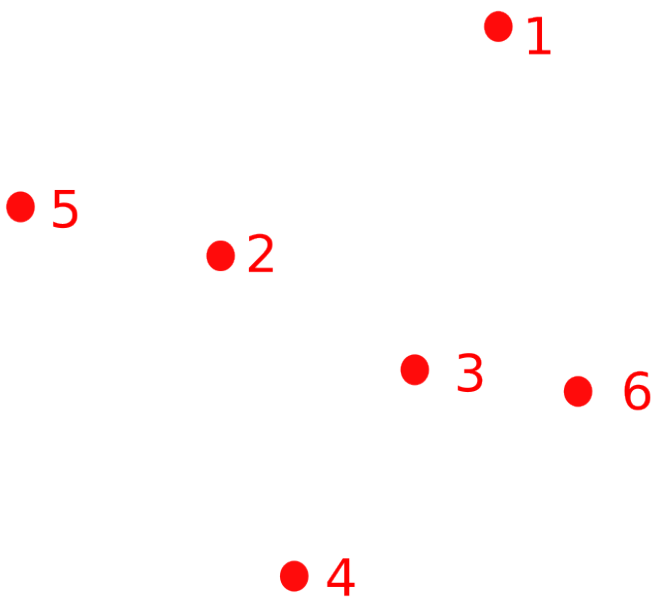
| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



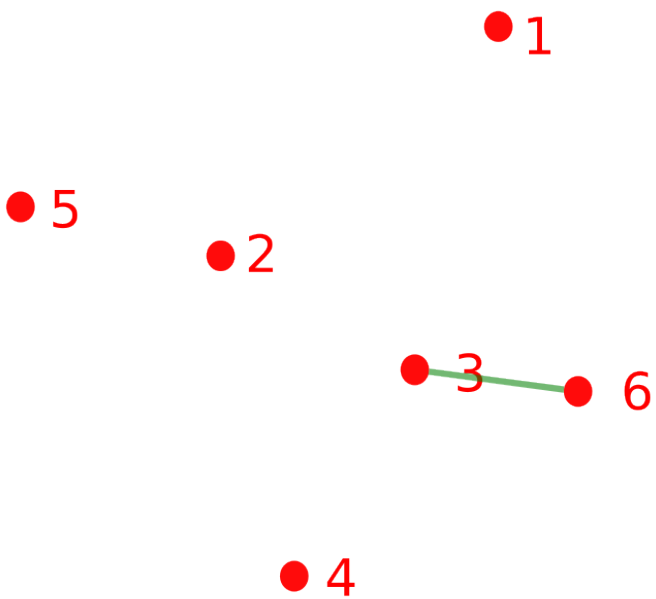
# Example - Complete Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



# Example - Complete Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |

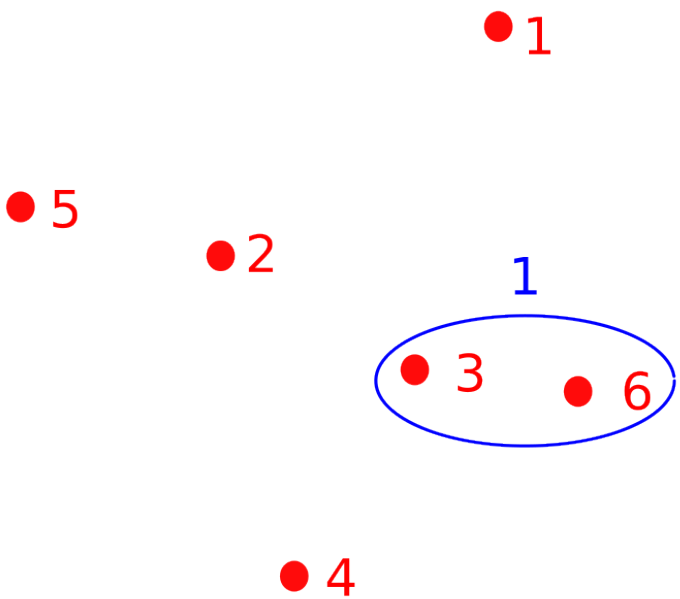



$$d(3,6) = 0.11$$

which is the minimum distance between points.

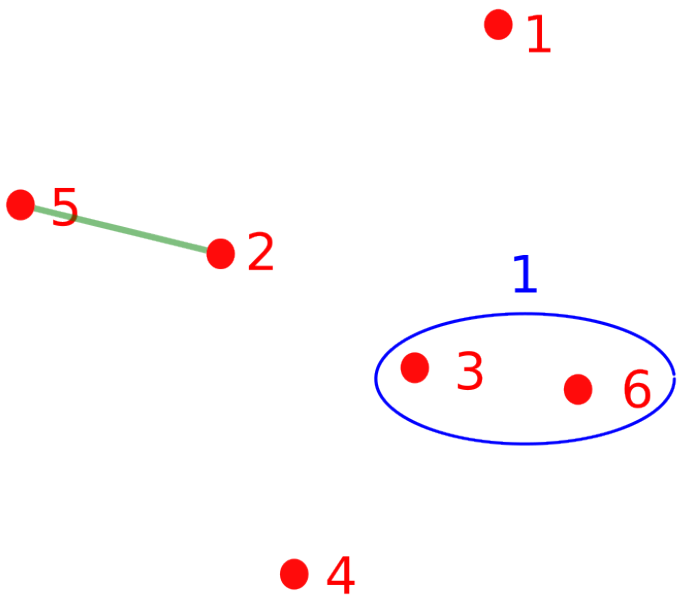
# Example - Complete Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



# Example - Complete Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |

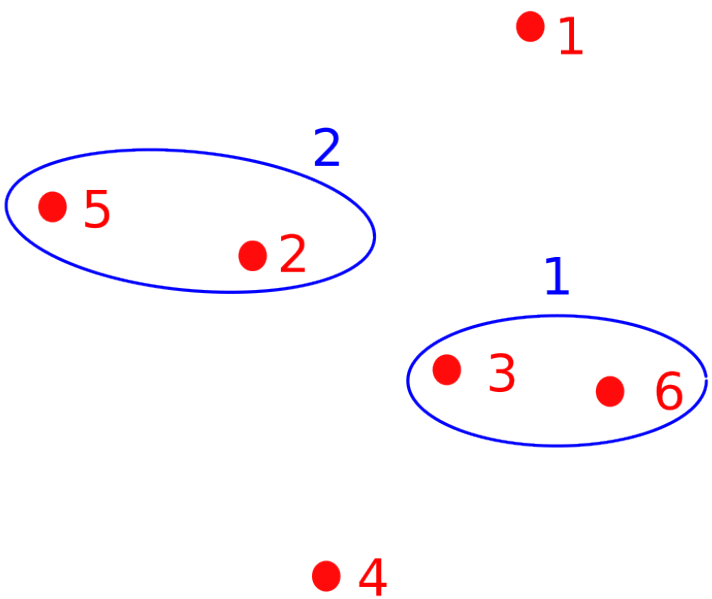



$d(2,5) = 0.14$

which is the second smallest distance.

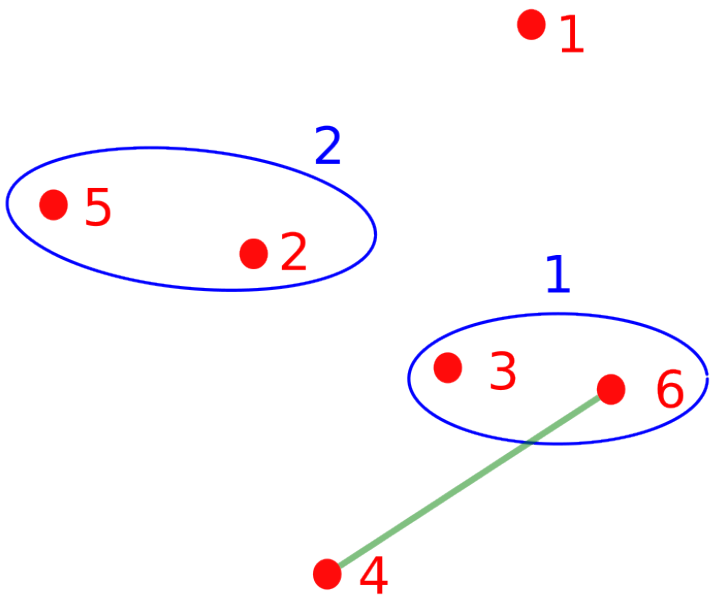
# Example - Complete Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



# Example - Complete Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |

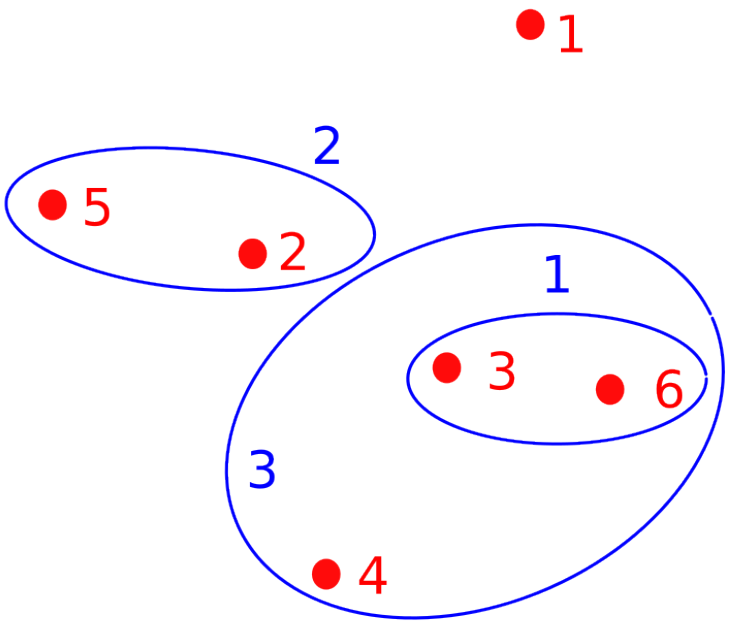



$$d(4,6) = 0.22 = \max\{d(4,3), d(4,6)\}$$

which is smaller than
$d(4,5) = 0.29$,
$d(1,5) = 0.34$,
$d(1,6) = 0.23$,
$d(5,6) = 0.39$,
$d(4,1) = 0.37$
the max distances of points in all other pairs of clusters.

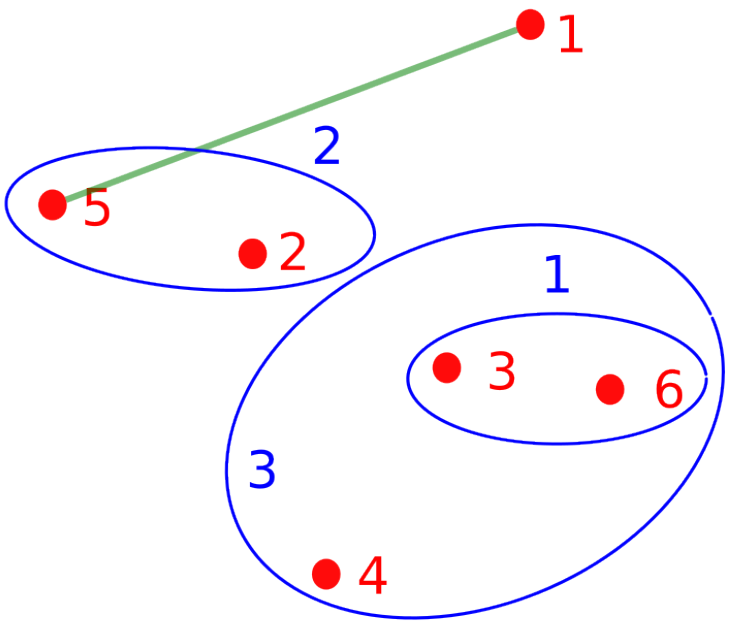
# Example - Complete Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



# Example - Complete Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |

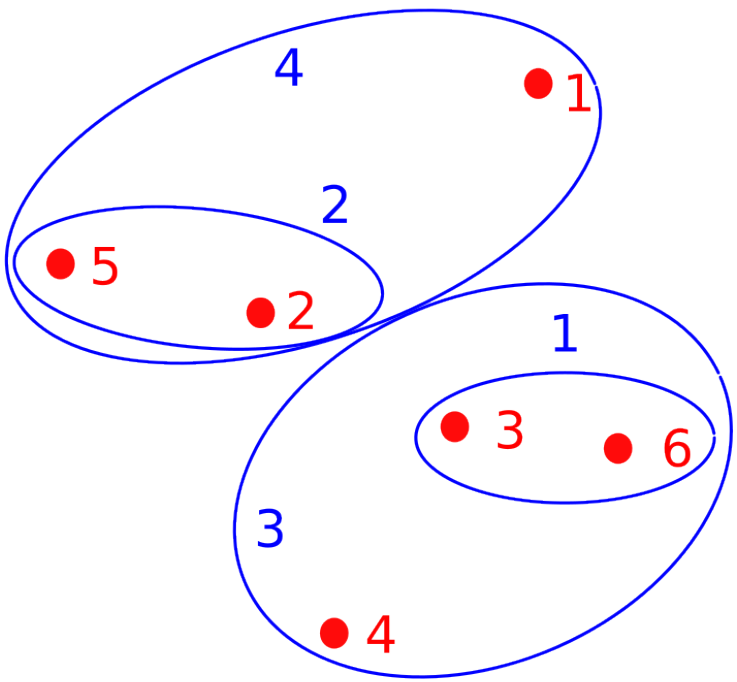



$$d(1,5) = 0.34$$

which is smaller than
$d(1,4) = 0.37$,
$d(5,6) = 0.39$,
which are the maximum distances of points in all other pairs of clusters.

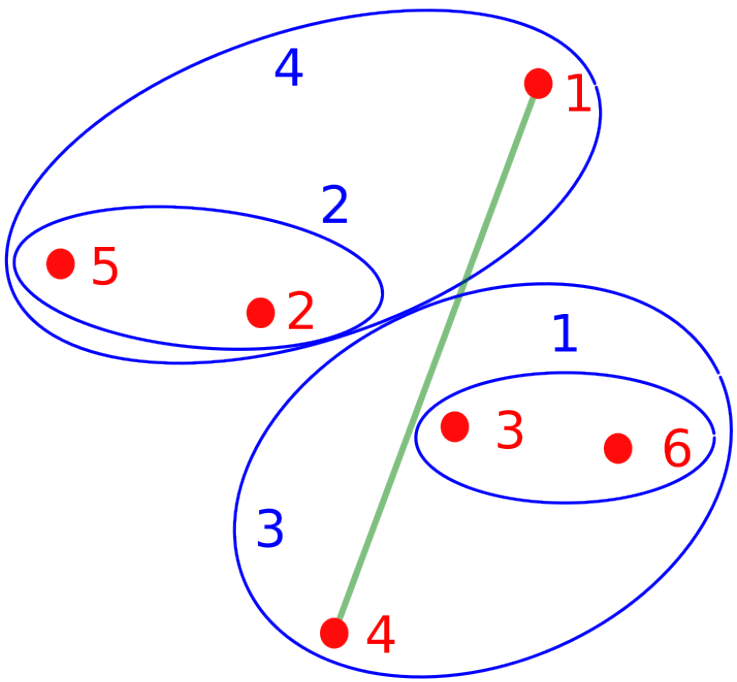
# Example - Complete Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



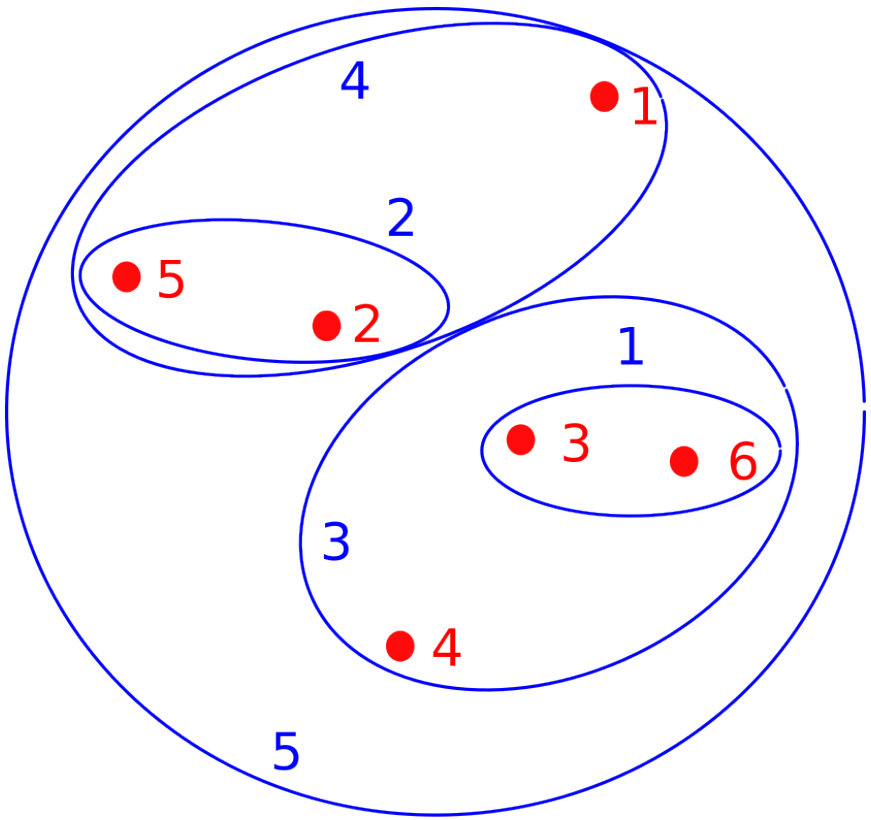
# Example - Complete Linkage

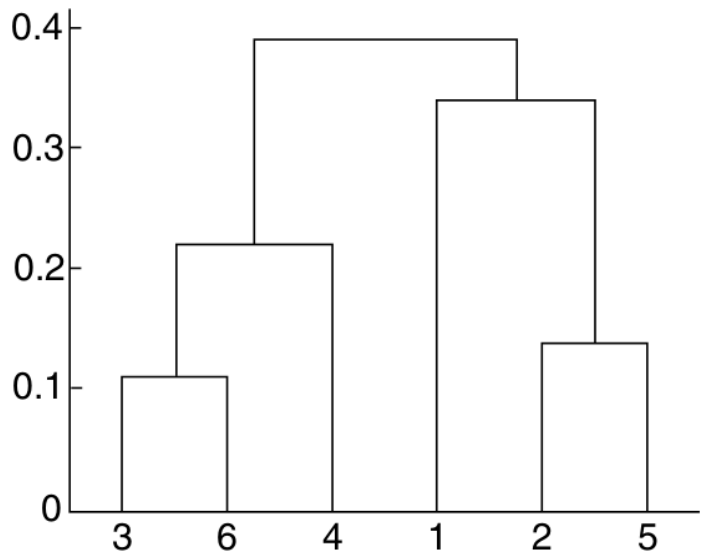
| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |



# Example - Complete Linkage

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.16 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.16 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |

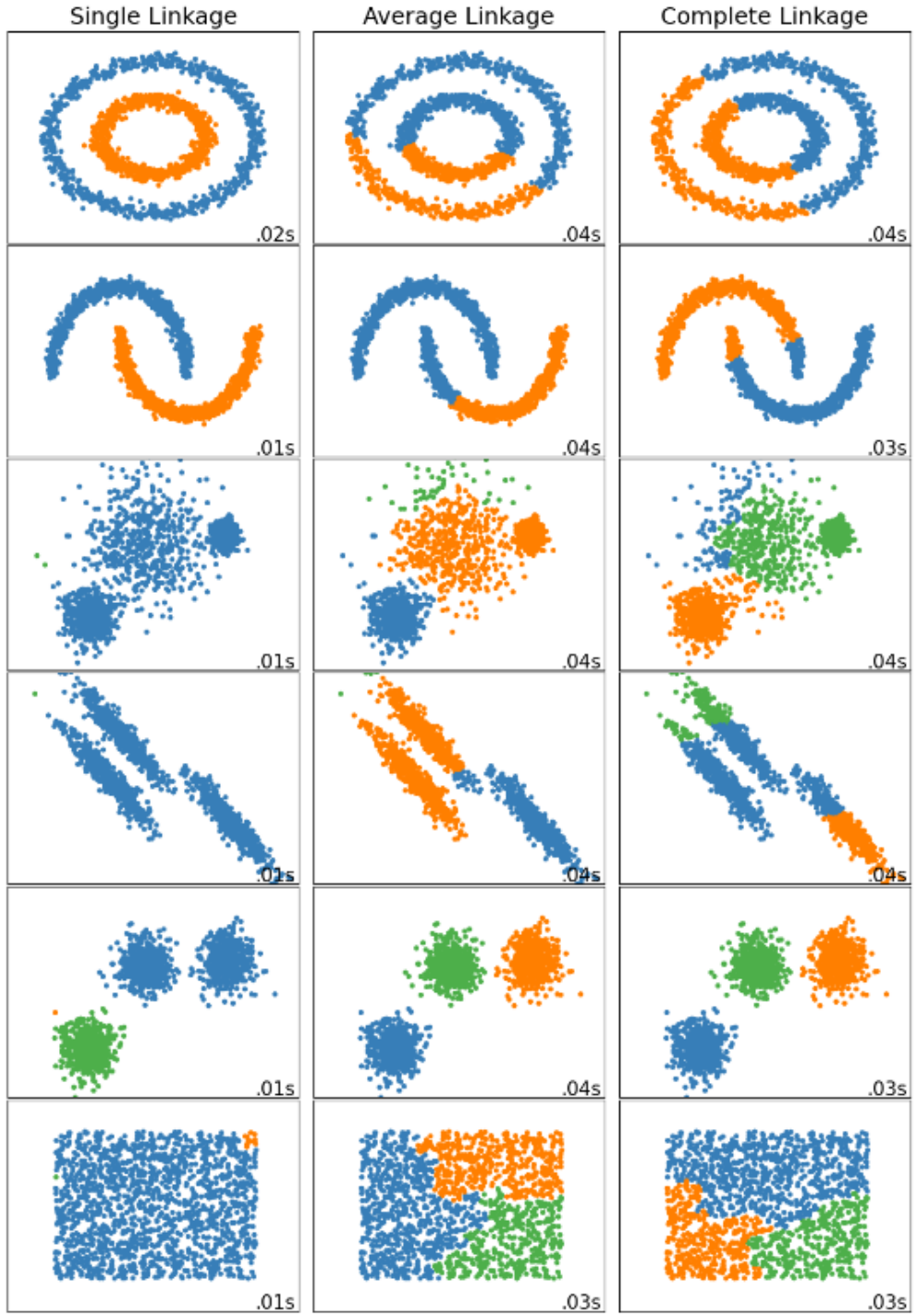


# Properties of Agglomerative Hierarchical Clustering

- Provides hierarchy of clusters - different cut levels provide different levels of coarseness of clusters
- Compared with $k$-means, it does not depend on the initialization and may provide better clusters than $k$-means.

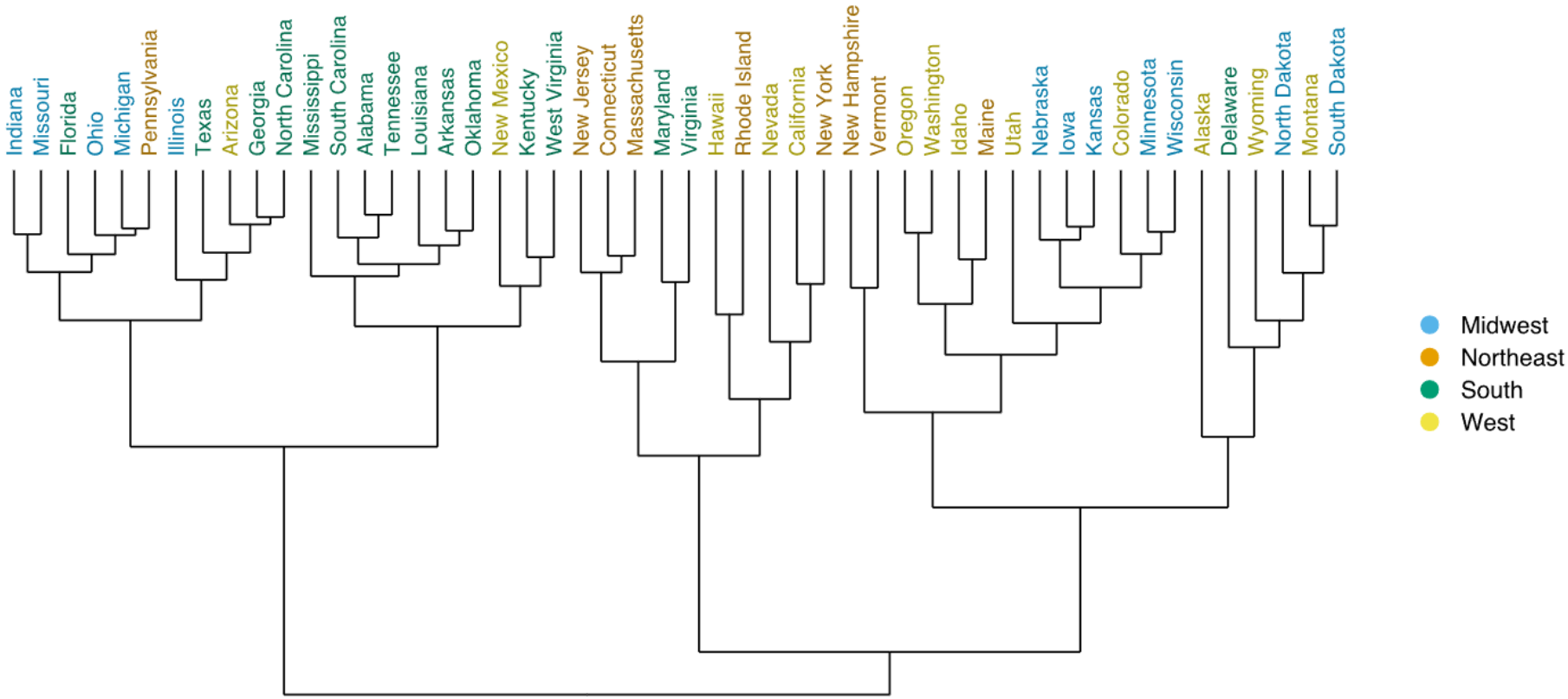
- Lack of global objective function
  - The agglomerative hierarchical clustering uses local criteria to decide which clusters to merge.
- Agglomerative clustering has a "rich get richer" behavior that leads to uneven cluster sizes
- Merging decision cannot be undone - bad for noisy data
- Computationally expensive.


Single Linkage
Average Linkage
Complete Linkage
.02s
.04s
.04s
.01s
.04s
.03s
.01s
.04s
.04s
.01s
.04s
.04s
.01s
.04s
.03s
.01s
.03s
.03s

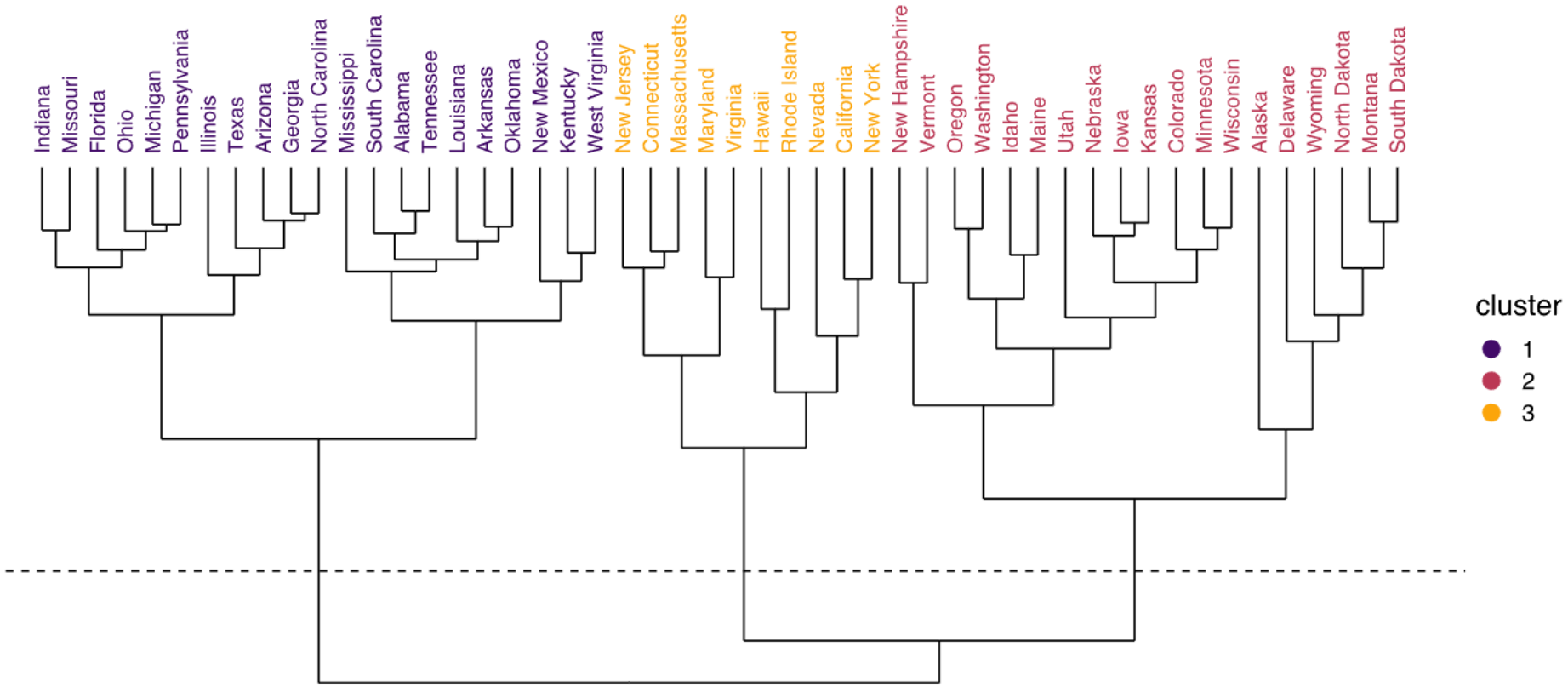
# State level statistics for US

```
# A tibble: 50 × 20
   state     homeo…¹ multi…² income med_i…³ poverty fed_s…⁴ smoke murder robbery
   <chr>       <dbl>   <dbl>  <dbl>   <dbl>   <dbl>   <dbl> <dbl>  <dbl>   <dbl>
 1 Alabama      71.1    15.5  22984   42081    17.1   11.7   24.8    8.2   141.
 2 Alaska       64.7    24.6  30726   66521     9.5   16.8   25      4.8    80.9
 3 Arizona      67.4    20.7  25680   50448    15.3    9.85  20.4    7.5   144.
 4 Arkansas     67.7    15.2  21274   39267    18      9.61  23.5    6.7    91.1
 5 Californ…    57.4    30.7  29188   60883    13.7    8.89  15.2    6.9   176.
 6 Colorado     67.6    25.6  30151   56456    12.2    9.15  19.9    3.7    84.6
 7 Connecti…    69.2    34.6  36775   67740     9.2   14.8   16.5    2.9   113
 8 Delaware     73.6    17.7  29007   57599    11      8.89  20.7    4.4   155.
 9 Florida      69.7    30    26551   47661    13.8    9.62  21.6    5     169.
10 Georgia      67.2    20.5  25134   49347    15.7    8.88  22.2    6.2   155.
# … with 40 more rows, 10 more variables: agg_assault <dbl>, larceny <dbl>,
#   motor_theft <dbl>, soc_sec <dbl>, nuclear <dbl>, coal <dbl>,
#   tr_deaths <dbl>, tr_deaths_no_alc <dbl>, unempl <dbl>, popdens2010 <dbl>,
#   and abbreviated variable names ¹homeownership, ²multiunit, ³med_income,
#   ⁴fed_spend
```

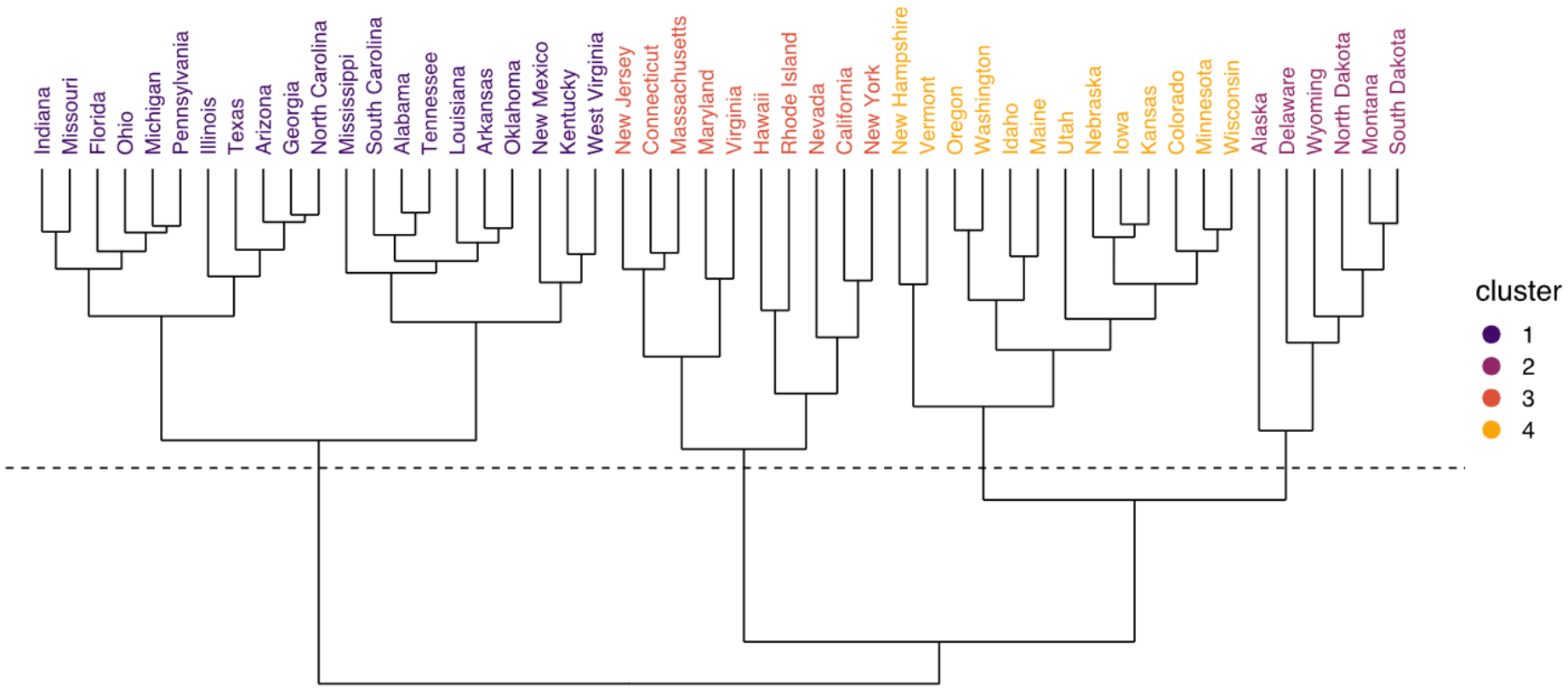
# State level statistics for US



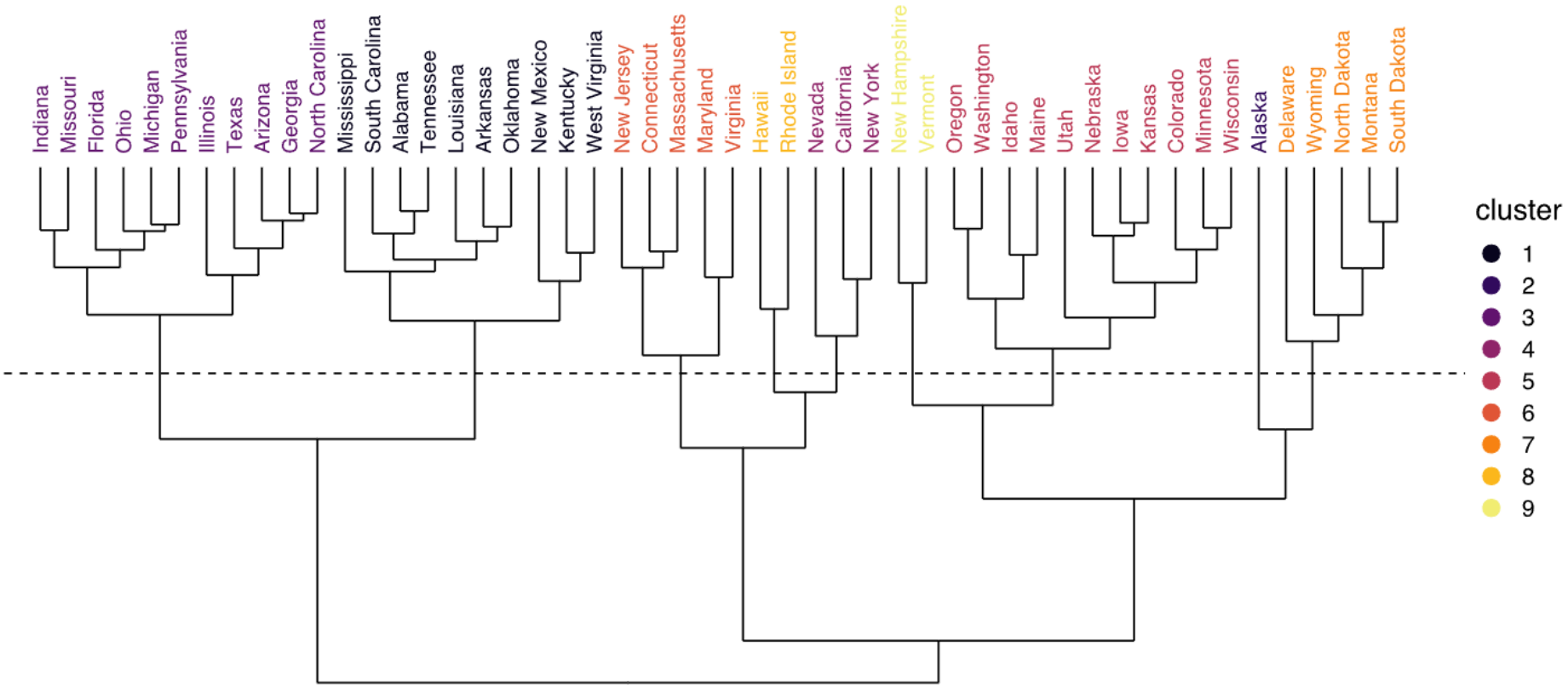
# State level statistics for US



# State level statistics for US



# State level statistics for US



# Cluster Validation

# Cluster Validity

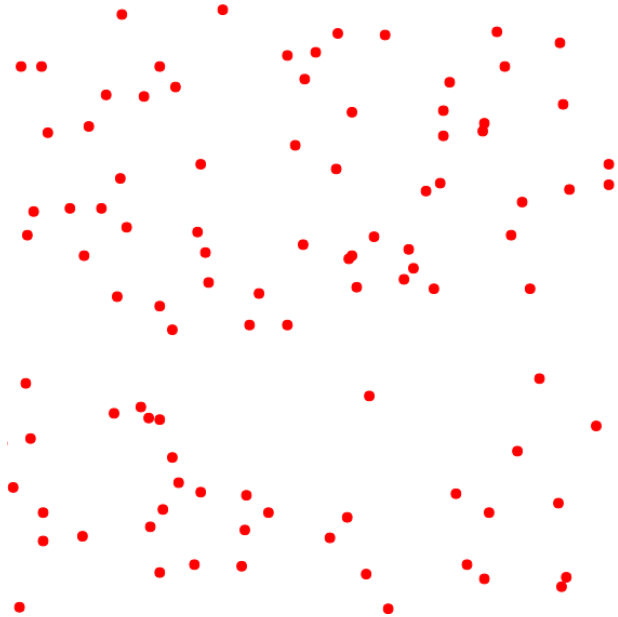
For supervised classification (= we have class labels) we have a variety of measures to evaluate how good our model is: Accuracy, Precision, Recall, $F_1$, etc.

For cluster analysis (=unsupervised learning), the analogous question is:

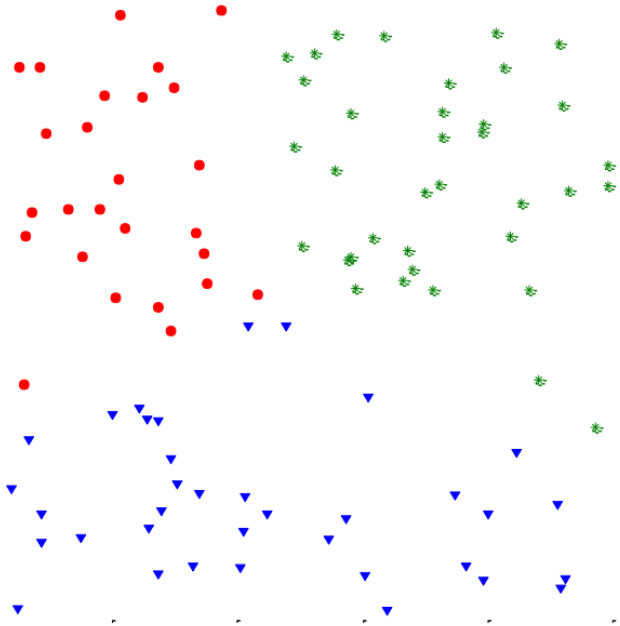
How to evaluate the "goodness" of the resulting clusters?

Keep in mind that the dataset can be large and high-dimensional. Visualization might be difficult.

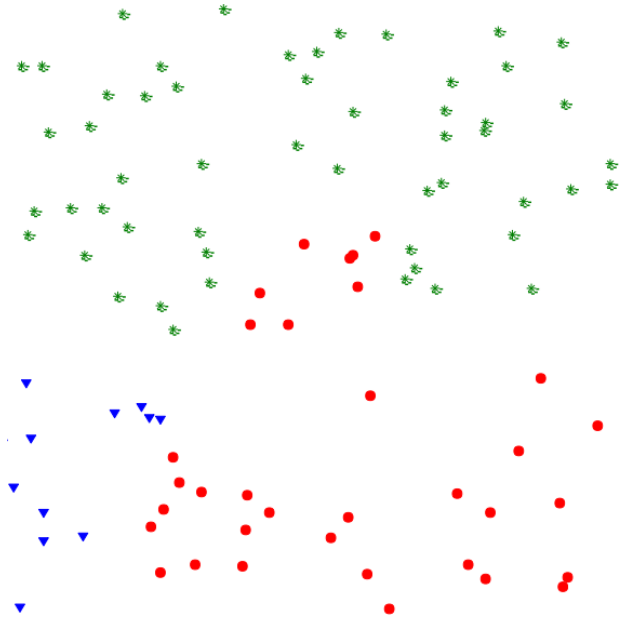
Random points:



K-means



Hierarchical



# Different Aspects of Cluster Validation

1. Determining the **clustering tendency** of a set of data, i.e., distinguishing whether non-random structure exists in the data (e.g., to avoid overfitting).
2. Internal Validation: Evaluating how well the cluster analysis results fit the data without reference to external information.
3. External Validation: Compare the cluster analysis results to externally known class labels (class labels).
4. Compare clusterings to determine which is better.
5. Determining the '**correct**' number of clusters.

For 2, 3, and 4, we can further distinguish whether we want to evaluate the entire clustering or just individual clusters.

# Measures of Cluster Validity

Numerical measures applied to judge various aspects of cluster validity are classified into the following three types.

- Internal Index: Used to measure the goodness of a clustering structure without respect to external information.
- External Index: Used to measure the extent to which cluster labels match externally supplied class labels.
- Relative Index: Used to compare two different clusterings or clusters.

# Internal Index

Consider a dataset

$$D = \{\vec{x}_1, \ldots, \vec{x}_p\}$$

Assume that a clustering algorithm produced a partition $\mathcal{U} = \{U_1, \ldots, U_K\}$ of $D$ into $K$ clusters.

No other information has been provided.

We aim to measure the clustering's "niceness" (??)

Assume that we have a distance measure $d$ measuring how far apart the objects being clustered.

**For concreteness**:

- We stick with numerical features, which means that the dataset $D = \{\vec{x}_1, \ldots, \vec{x}_p\}$ contains vectors $\vec{x}_i \in \mathbb{R}^n$.
- Assume the Euclidean distance $d$.

Note that the validity measures may be based on completely different similarity/dissimilarity measures and non-numerical data.

# Cohesion and Separation

Consider a dataset $D = \{\vec{x}_1, \ldots, \vec{x}_p\}$ and its clustering $\mathcal{U} = \{U_1, \ldots, U_K\}$.

Let us utilize the concept of distance to cluster prototypes and consider the distance between prototypes.

To measure the dissimilarity of examples, we use the notion of *proximity* defined on pairs of vectors $\vec{x}, \vec{z}$:

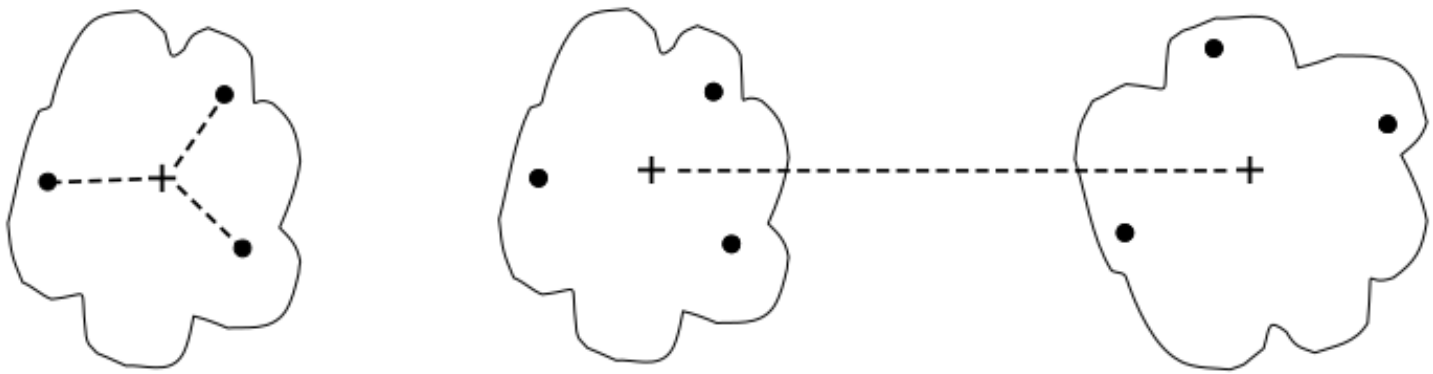
$$proximity(\vec{x}, \vec{z})$$

The proximity might be, e.g.,

- the distance $d(\vec{x}, \vec{z})$,
- the square of the distance, that is $d(\vec{x}, \vec{z})^2$,
- any other notion of dissimilarity based on the application.

We consider the notions of *cohesion* (proximity of examples within clusters) and *separation* (proximity of clusters).

# Prototype-based Cohesion and Separation



Prototype-based cohesion = the similarity of examples within a given cluster to a prototype of the cluster (e.g., centroid).

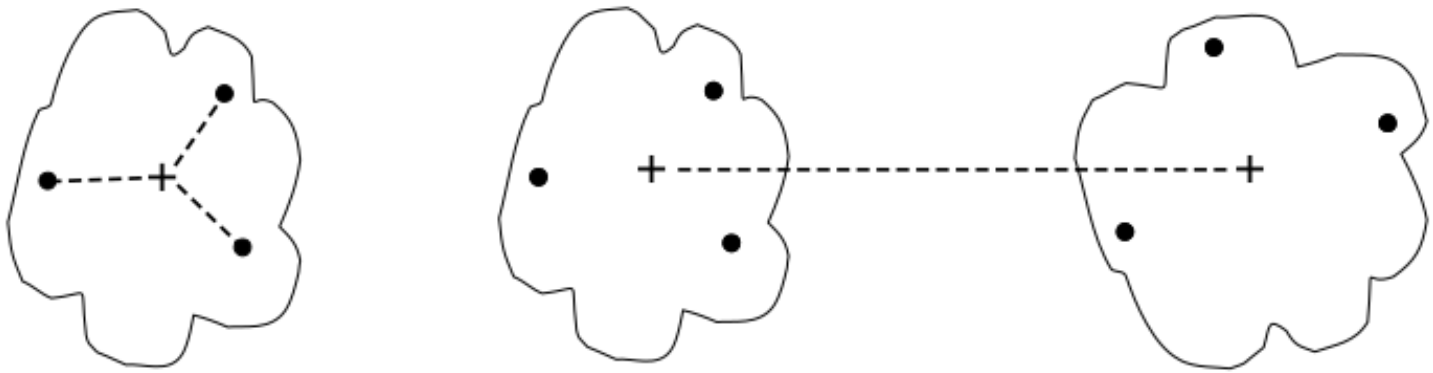
Given a cluster $U_j \in \mathcal{U}$ and its prototype $\vec{m}_j \in \mathbb{R}^n$,

$$cohesion(U_j) = \sum_{\vec{x} \in U_j} proximity(\vec{x}, \vec{m}_j)$$

Note that the prototype **does not** have to be an element of $U_j$.

Intuitively, cohesion is the proximity of cluster's examples and a point somewhere "between" all examples of the cluster.

# Prototype-based Cohesion and Separation



Prototype-based separation = dissimilarity of prototypes of different clusters.

Given a cluster $U_j \in \mathcal{U}$, its prototype $\vec{m}_j \in \mathbb{R}^n$, and a prototype of all examples $\vec{m} \in \mathbb{R}^n$ (e.g. the centroid of all examples)

$$\mathit{separation}(U_j) = \mathit{proximity}(\vec{m}_j, \vec{m})$$

Intuitively, separation is the proximity of the cluster's examples to the dataset's center.

# Prototype-based Cohesion and Separation

Summarize the prototype-based cohesion and separation as follows:

$$\begin{aligned}\text{cohesion}(\mathcal{U}) &= \sum_{j=1}^{K} \mathit{cohesion}(U_j) \\ &= \sum_{j=1}^{K} \sum_{\vec{x} \in U_j} \mathit{proximity}(\vec{x}, \vec{m}_j)\end{aligned}$$

$$\begin{aligned}\text{separation}(\mathcal{U}) &= \sum_{j=1}^{K} |U_j| \mathit{separation}(U_j) \\ &= \sum_{j=1}^{K} |U_j| \mathit{proximity}(\vec{m}_j, \vec{m})\end{aligned}$$

If $\mathit{proximity}(\vec{x}, \vec{z})$ is defined as $d(\vec{x}, \vec{z})^2$ then the cohesion is the inertia.

There is an interesting relationship between the above measures and the squared distances to the prototype of the whole dataset $\vec{m}$.

# Prototype-based Cohesion and Separation

Consider a dataset $D = \{\vec{x}_1, \ldots, \vec{x}_p\}$ and its clustering $\mathcal{U} = \{U_1, \ldots, U_K\}$ of $D$.

Consider $\mathit{proximity}(\vec{x}, \vec{z}) = d(\vec{x}, \vec{z})^2$ and all prototypes to be centroids.

Let $\vec{m} \in \mathbb{R}^n$ be the centroid of all examples:

$$\vec{m} = \frac{1}{|D|} \sum_{i=1}^{p} \vec{x}_i$$

Define

$$\text{TSS} = \sum_{i=1}^{p} \mathit{proximity}(\vec{x}_i, \vec{m}) = \sum_{i=1}^{p} d(\vec{x}_i, \vec{m})^2$$

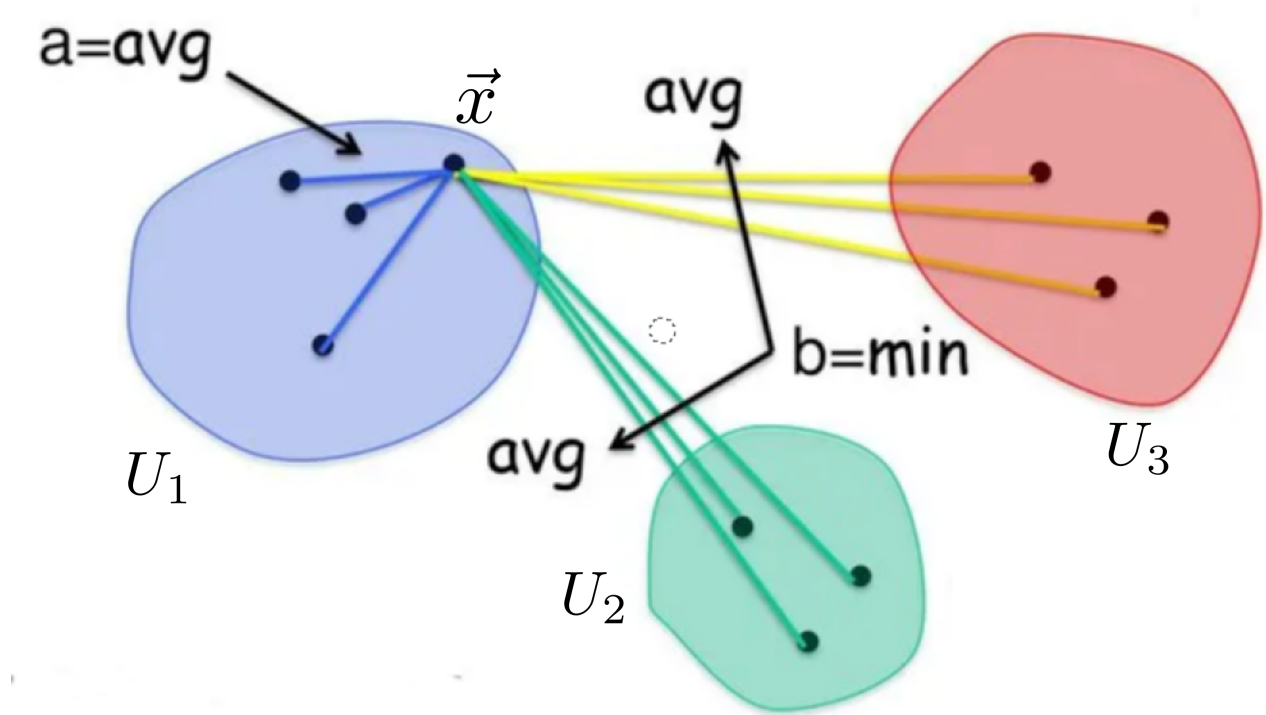
The following holds:

$$\text{TSS} = \text{cohesion}(\mathcal{U}) + \text{separation}(\mathcal{U})$$

Note that TSS is determined by $D$.

# Silouette Score

*Silhouette score* can be used to measure both qualities of clustering from the point of view of individual examples and from the point of view of the overall clustering.




$$\text{silhouette}(\vec{x}) = \frac{b - a}{\max\{a, b\}}$$

# Silhouette Score

Consider a clustering $\mathcal{U} = \{U_1, \ldots, U_k\}$ and $\vec{x} \in U_j$.

If $|U_j| > 1$ we define

$$a(\vec{x}) = \frac{1}{|U_j| - 1} \sum_{\vec{z} \in U_j \setminus \{\vec{x}\}} d(\vec{x}, \vec{z})$$

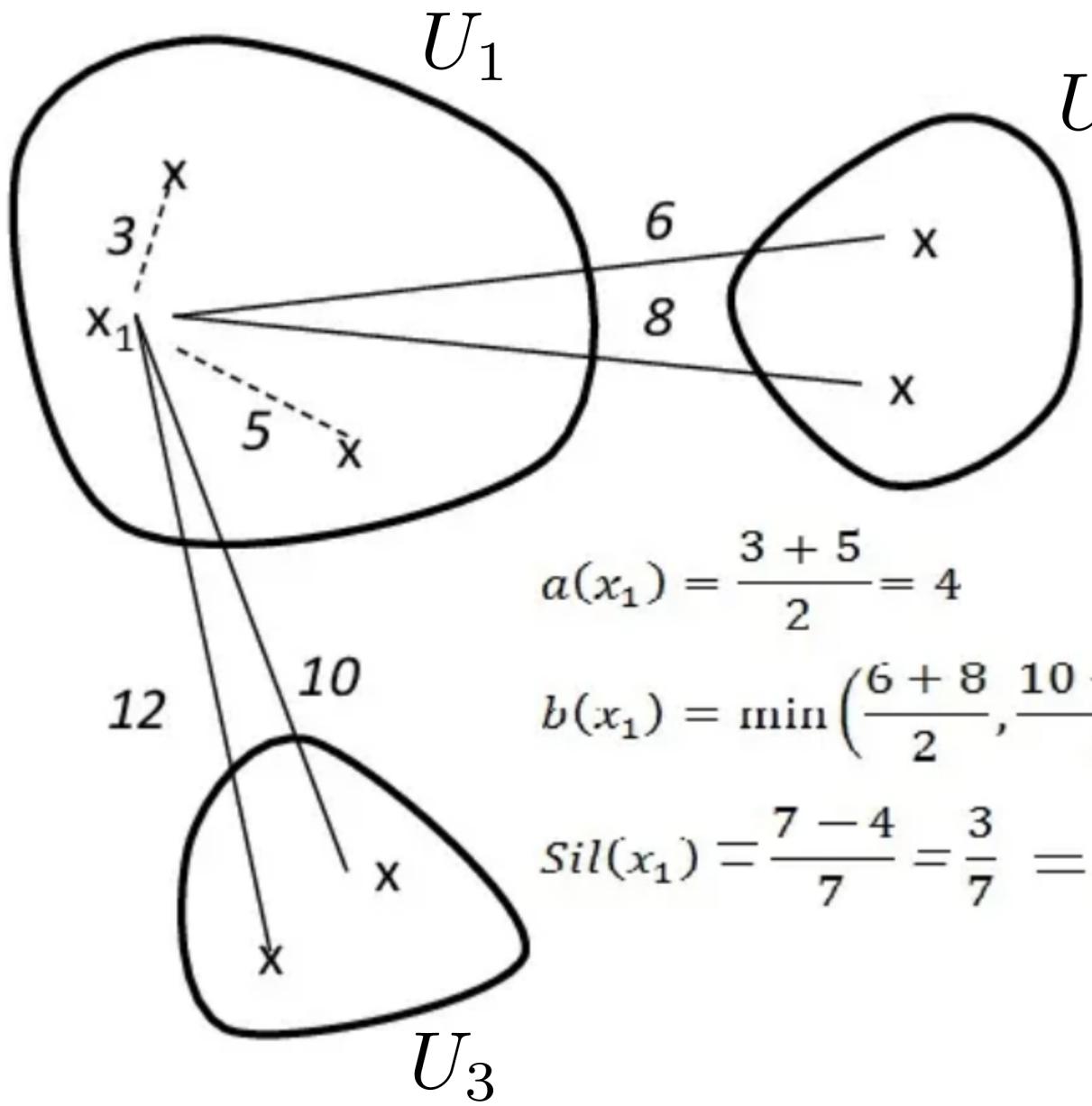
$$b(\vec{x}) = \min_{k \neq j} \frac{1}{|U_k|} \sum_{\vec{z} \in U_k} d(\vec{x}, \vec{z})$$

If $|U_j| > 1$ we define

$$\mathit{silhouette}(\vec{x}) = \frac{b(\vec{x}) - a(\vec{x})}{\max\{a(\vec{x}), b(\vec{x})\}}$$

Else, we define $\mathit{silhouette}(\vec{x}) = 0$.

# Example

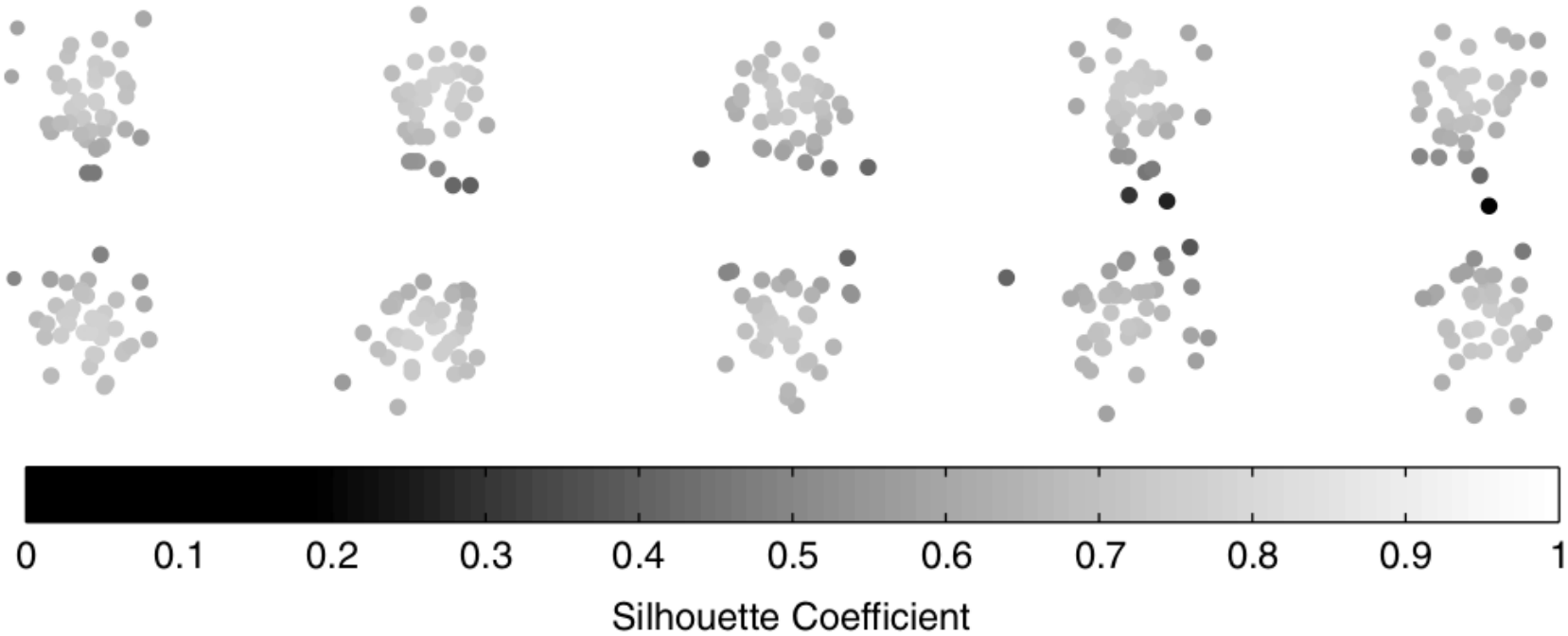



$$a(x_1) = \frac{3+5}{2} = 4$$

$$b(x_1) = \min\left(\frac{6+8}{2}, \frac{10+12}{2}\right) = 7$$

$$Sil(x_1) = \frac{7-4}{7} = \frac{3}{7} = 0.42$$

# Example



# Silhouette for Clusters and Clusterings

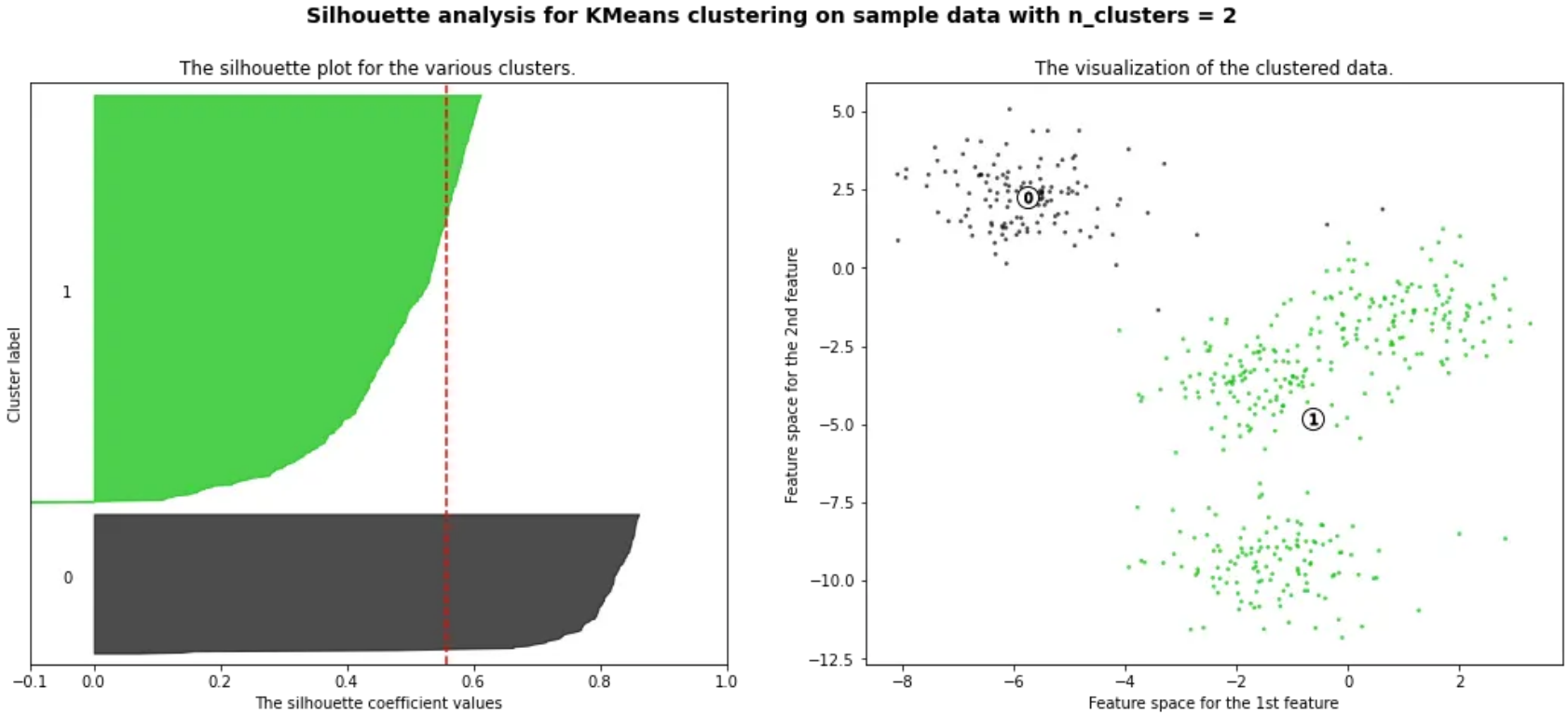
We have defined the silhouette for a single $\vec{x} \in D$.

To obtain the silhouette score for a whole cluster $U_j$ or for $D$ we summarize using simple averaging:

$$silhouette(U_j) = \frac{1}{|U_j|} \sum_{\vec{x} \in U_j} silhouette(\vec{x})$$

$$silhouette(D) = \frac{1}{|D|} \sum_{\vec{x} \in D} silhouette(\vec{x})$$

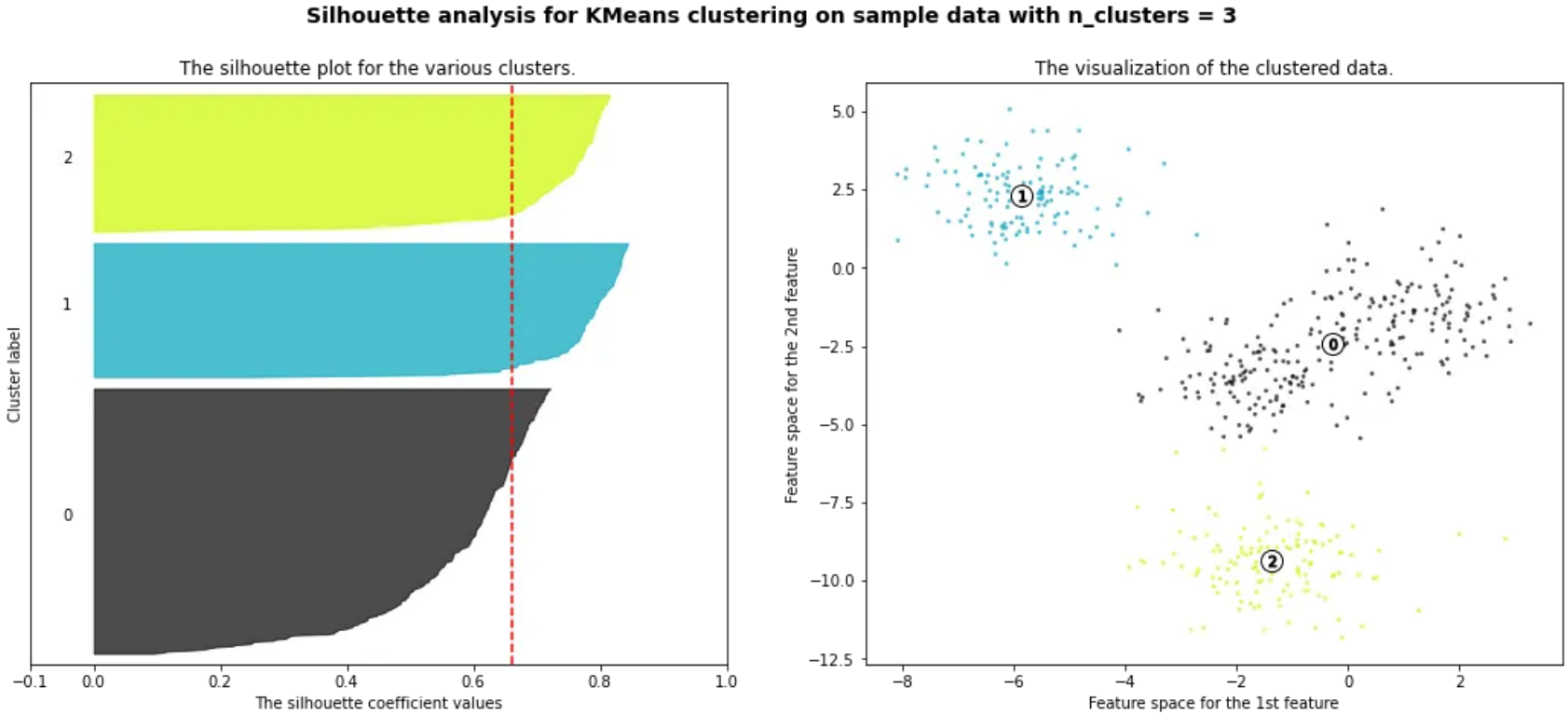
# Example




The colored graphs on the left are silhouette scores of the individual elements of clusters.

The red vertical line denotes the silhouette score for the whole dataset.

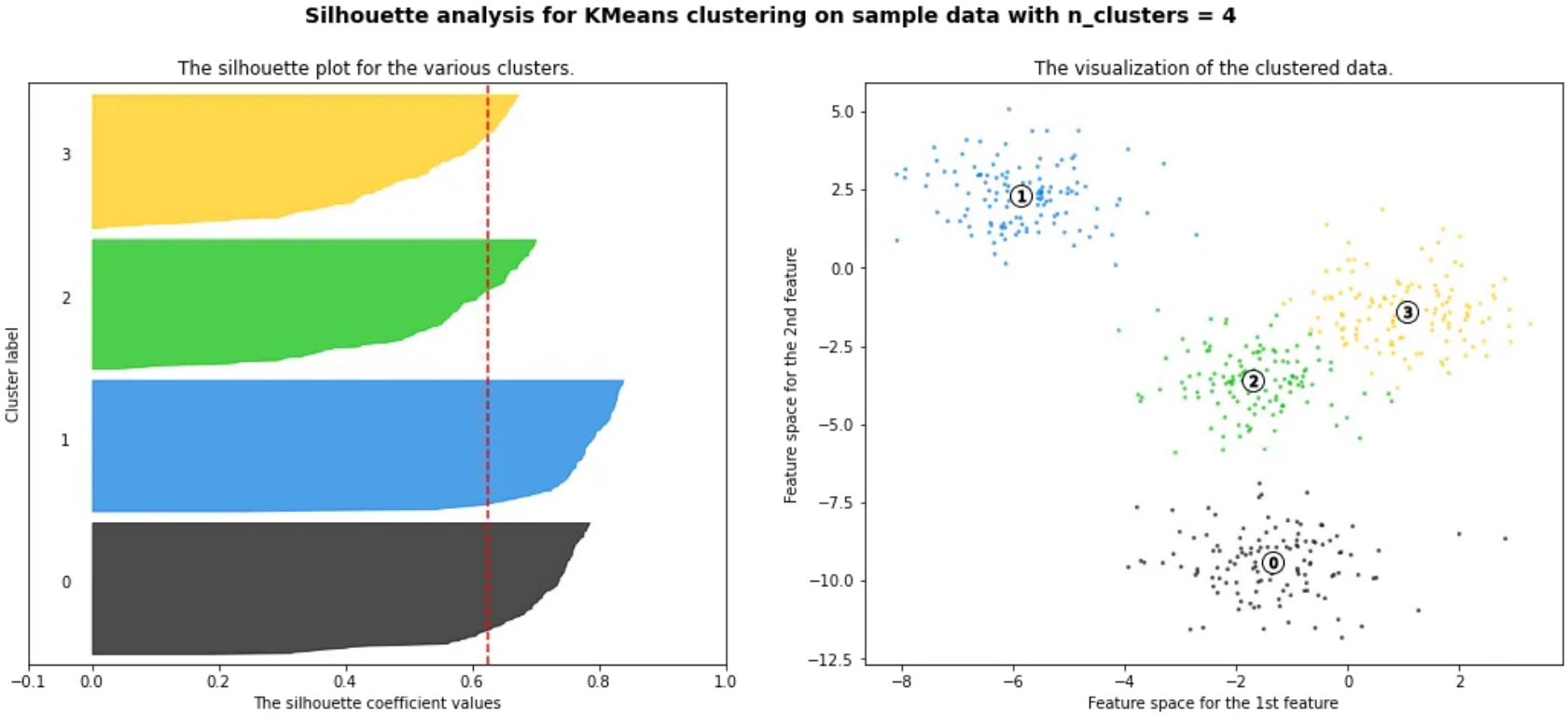
# Example




The colored graphs on the left are silhouette scores of the individual elements of clusters.

The red vertical line denotes the silhouette score for the whole dataset.

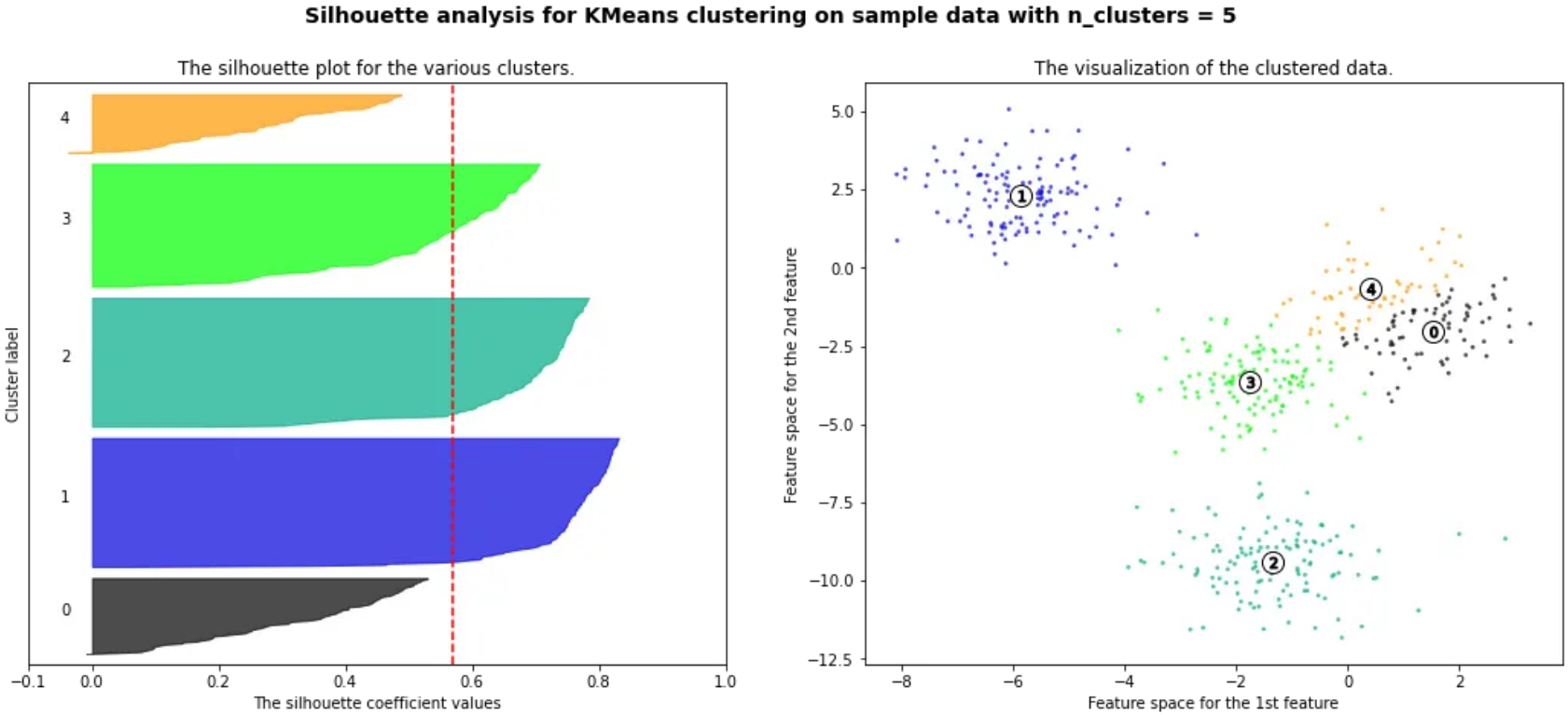
# Example




The colored graphs on the left are silhouette scores of the individual elements of clusters.

The red vertical line denotes the silhouette score for the whole dataset.

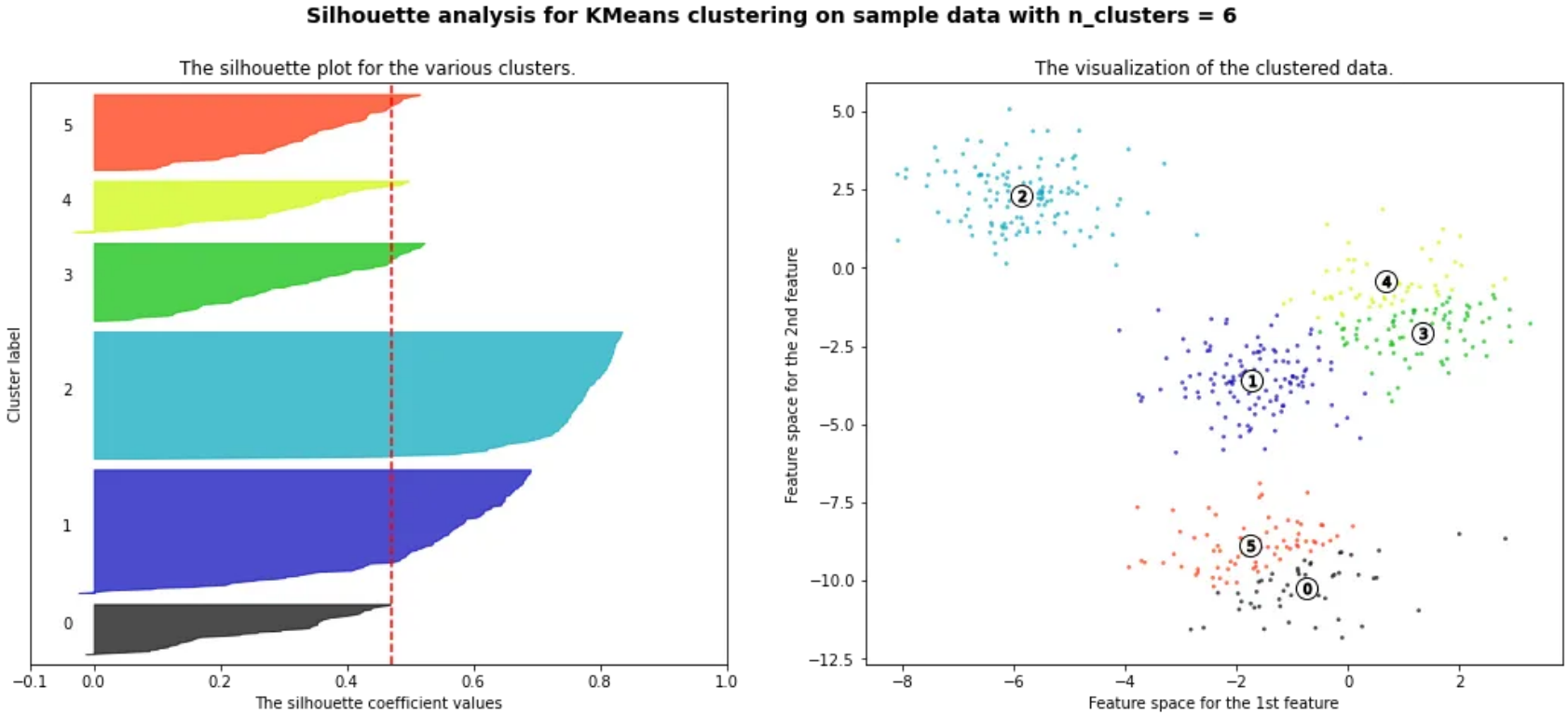
# Example




The colored graphs on the left are silhouette scores of the individual elements of clusters.

The red vertical line denotes the silhouette score for the whole dataset.

# Example




The colored graphs on the left are silhouette scores of the individual elements of clusters.

The red vertical line denotes the silhouette score for the whole dataset.

# External Index

Consider a *supervised learning* dataset

$$D = \{(\vec{x}_1, c_1), \ldots, (\vec{x}_p, c_p)\}$$

Here $c_i \in C$ is a class of $\vec{x}_i$.

Assume that a clustering algorithm produced a partition $\mathcal{U} = \{U_1, \ldots, U_K\}$ of $D$ into $K$ clusters.

We measure how the clustering conforms with the given classes.

# Purity

Consider the clustering to be a classification model.

Define a classifier $h : D \to C$ such that given $\vec{x_i} \in U_j \in \mathcal{U}$

$$h(\vec{x_i}) = \text{the most frequent class in } U_j$$

Now we can measure the Accuracy of $h$.

Accuracy of $h$ is called *purity*.

Intuitively, it is the proportion of non-majority class elements in clusters.

Is it a good measure?

Probably not; many clusters lead to high purity (each element in its own cluster means purity $= 1$).

# Classifier Point of View

Given $\vec{x}_i$, denote by $\mathcal{U}(\vec{x}_i)$ the cluster $U_j \in \mathcal{U}$ containing $\vec{x}_i$.

Distinguish the following types of *pairs* of examples:

- TP = number of examples of the same class and the same cluster

$$\mathrm{TP} = |\{(i,j) \mid \mathcal{U}(\vec{x}_i) = \mathcal{U}(\vec{x}_j) \wedge c_i = c_j)\}$$

- TN = number of examples of different classes and different clusters

$$\mathrm{TN} = |\{(i,j) \mid \mathcal{U}(\vec{x}_i) \neq \mathcal{U}(\vec{x}_j) \wedge c_i \neq c_j)\}$$

- FP = number of examples of different classes and the same cluster

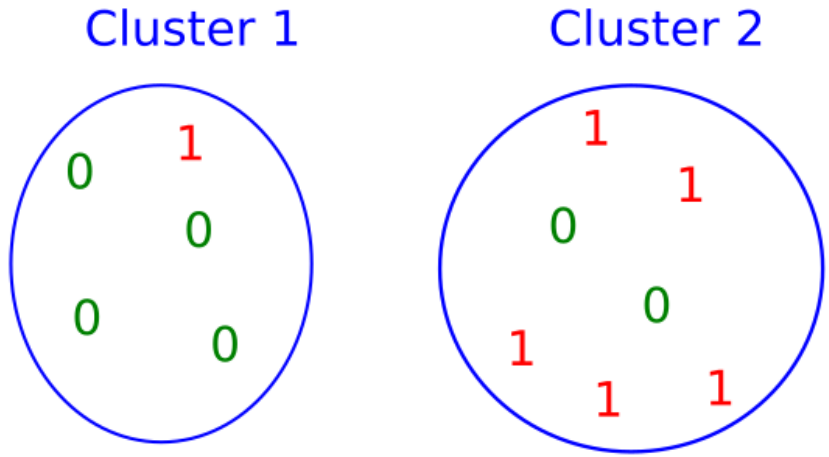
$$\mathrm{FP} = |\{(i,j) \mid \mathcal{U}(\vec{x}_i) = \mathcal{U}(\vec{x}_j) \wedge c_i \neq c_j)\}$$

- FN = number of examples of the same class and different clusters

$$\mathrm{FN} = |\{(i,j) \mid \mathcal{U}(\vec{x}_i) \neq \mathcal{U}(\vec{x}_j) \wedge c_i = c_j)\}$$

Now, we may apply all the measures from the supervised model.

# Example




$$TP = \binom{4}{2} + \binom{5}{2} + \binom{2}{2}$$
$$= 6 + 10 + 1 = 17$$

$$TN = 4 * 5 + 1 * 2 = 22$$

$$FP = 1 * 4 + 5 * 2 = 14$$

$$FN = 1 * 5 + 4 * 2 = 13$$

| | | Cluster | |
|---|---|---|---|
| | | *same* | *diff* |
| Class | *same* | TP=17 | FN=13 |
| | *diff* | FP=14 | TN=22 |

# Rand Index

Accuracy (in this area known as *Rand index*) is

$$\text{RandInd} = \text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

In our example,

$$\text{RandInd} = (17 + 22)/(17 + 22 + 14 + 13) = 0.59$$

Here, note that the Rand index considers the purity and the number of clusters.

Note that the Rand index can be used to compare two clusterings: Simply consider class labels to be indicators of clusters.

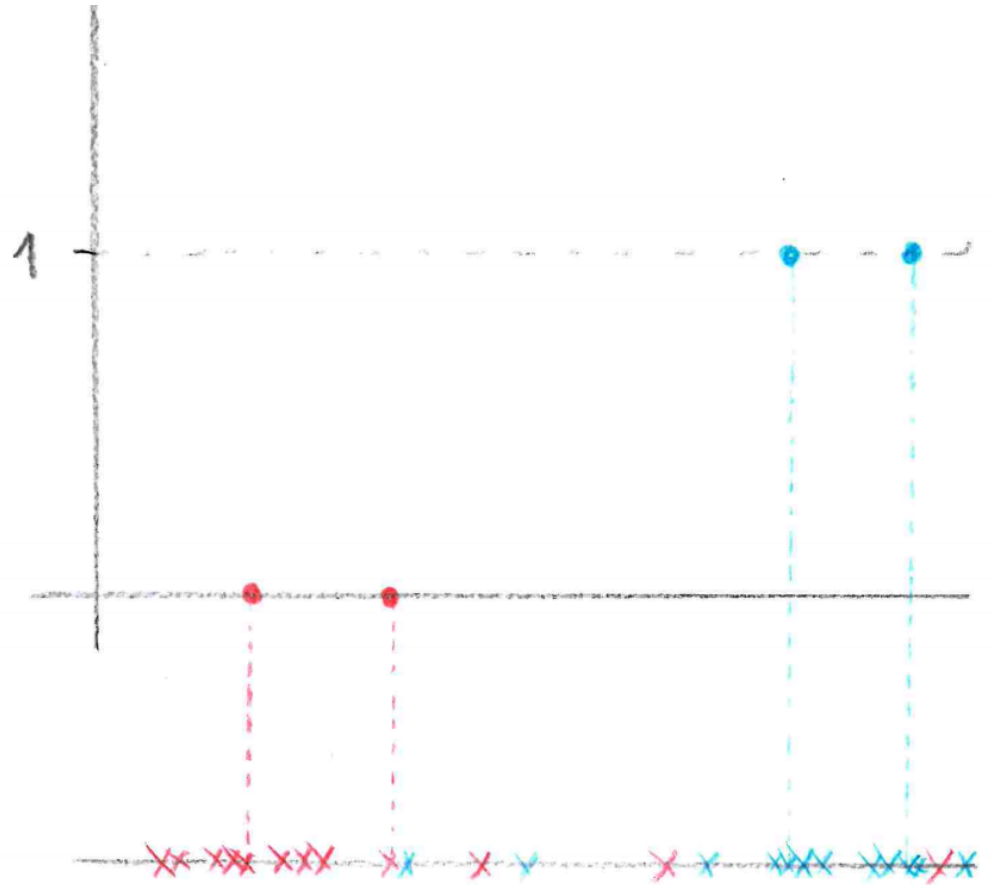
Similarly, we may compute the other measures such as Precision, Recall, and $F_1$ with all their benefits and limitations.

# Logistic Regression

# What about classification using regression?

Binary classification: Desired outputs 0 and 1

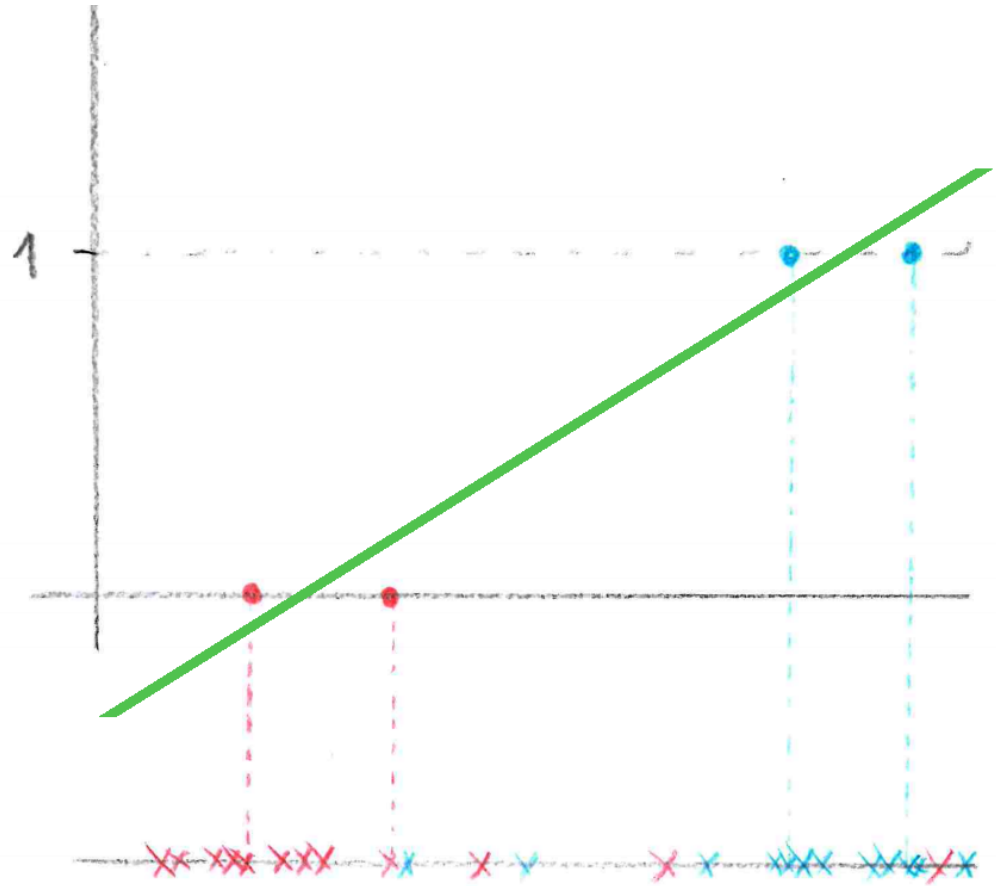
... we want to capture the probability distribution of the classes



# What about classification using regression?

Binary classification: Desired outputs 0 and 1

... we want to capture the probability distribution of the classes

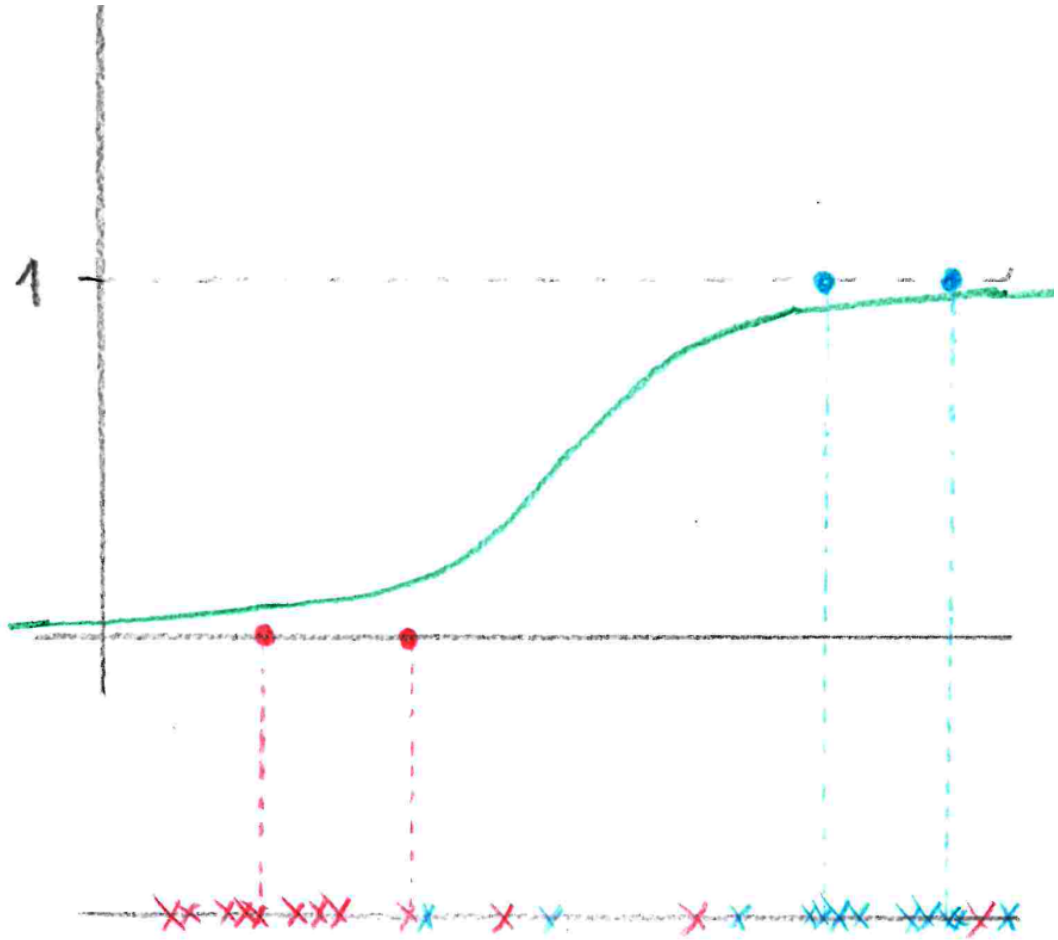



... does not capture the probability well (it is not probability at all)

# What about classification using regression?

Binary classification: Desired outputs 0 and 1

... we want to capture the probability distribution of the classes




... logistic sigmoid $\frac{1}{1+e^{-(\vec{w}\cdot\tilde{\mathbf{x}})}}$ is much better!

# Logistic Regression

**Logistic regression** model $h[\vec{w}]$ is determined by a vector of weights $\vec{w} = (w_0, w_1, \ldots, w_n) \in \mathbb{R}^{n+1}$ as follows:

Given $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$,

$$h[\vec{w}](\vec{x}) := \frac{1}{1 + e^{-\left(w_0 + \sum_{k=1}^{n} w_k x_k\right)}} = \frac{1}{1 + e^{-\vec{w} \cdot \tilde{\mathbf{x}}}}$$

Here

$$\tilde{\mathbf{x}} = (x_0, x_1, \ldots, x_n) \quad \text{where } x_0 = 1$$

is the *augmented feature vector*.

# But what is the meaning of the sigmoid?

The model gives probability $h[\vec{w}](\vec{x})$ of the class 1 given an input $\vec{x}$.
But why do we model such probability using $1/(1 + e^{-\vec{w}\cdot\tilde{\mathbf{x}}})$ ??

# But what is the meaning of the sigmoid?

The model gives probability $h[\vec{w}](\vec{x})$ of the class 1 given an input $\vec{x}$. But why do we model such probability using $1/(1 + e^{-\vec{w}\cdot\tilde{\mathbf{x}}})$ ??

Denote by $\bar{h}$ the probability $P(Y = 1 \mid X = \vec{x})$, i.e., the "true" probability of the class 1 given features $\vec{x}$.

---

The probability $\bar{h}$ cannot be easily modeled using a linear function (the probabilities are between 0 and 1).

# But what is the meaning of the sigmoid?

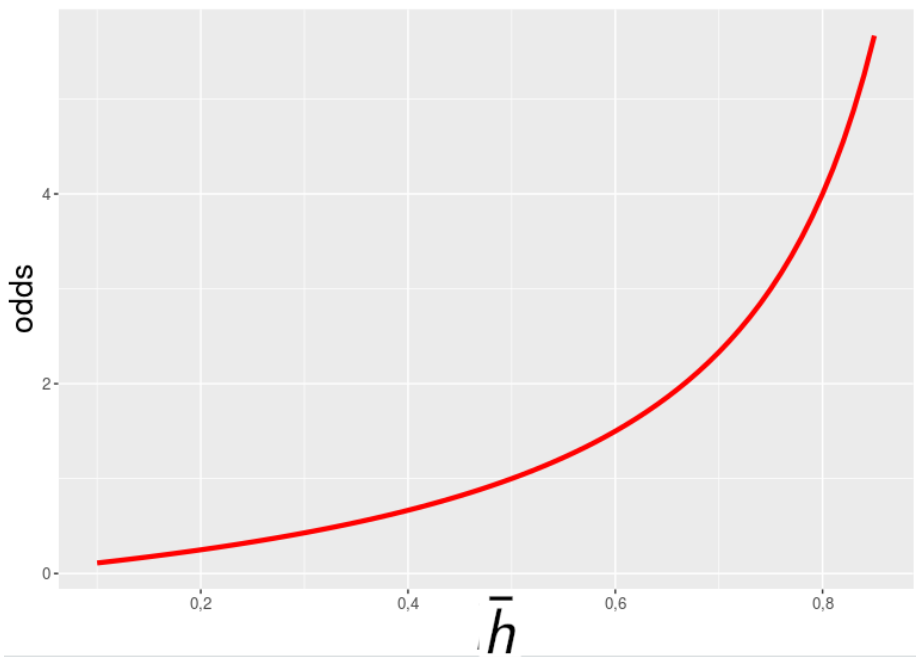
The model gives probability $h[\vec{w}](\vec{x})$ of the class 1 given an input $\vec{x}$. But why do we model such probability using $1/(1 + e^{-\vec{w}\cdot\tilde{\mathbf{x}}})$ ??

Denote by $\bar{h}$ the probability $P(Y = 1 \mid X = \vec{x})$, i.e., the "true" probability of the class 1 given features $\vec{x}$.

---

What about odds of the class 1?

$$odds(\bar{h}) = \bar{h}/(1 - \bar{h})$$




Better, at least it is unbounded on one side ...

# But what is the meaning of the sigmoid?

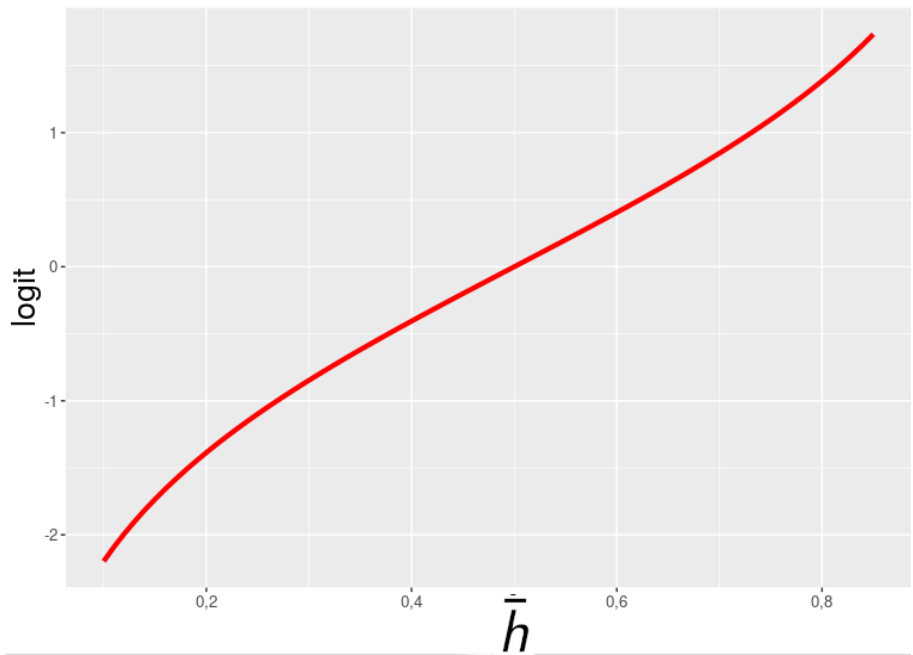
The model gives probability $h[\vec{w}](\vec{x})$ of the class 1 given an input $\vec{x}$. But why do we model such probability using $1/(1 + e^{-\vec{w}\cdot\tilde{\mathbf{x}}})$ ??

Denote by $\bar{h}$ the probability $P(Y = 1 \mid X = \vec{x})$, i.e., the "true" probability of the class 1 given features $\vec{x}$.

---

What about log odds (aka logit) of the class 1?

$$logit(\bar{h}) = \log(\bar{h}/(1 - \bar{h}))$$




Looks almost linear, at least for probabilities not too close to 0 or 1 ...

# But what is the meaning of the sigmoid?

Assume that $\bar{h}$ is the actual probability of the class 1 for an "object" with features $\vec{x} \in \mathbb{R}^n$. Put

$$\log(\bar{h}/(1-\bar{h})) = \vec{w} \cdot \tilde{\mathbf{x}}$$

Then

$$\log((1-\bar{h})/\bar{h})) = -\vec{w} \cdot \tilde{\mathbf{x}}$$

and

$$(1-\bar{h})/\bar{h} = e^{-\vec{w} \cdot \tilde{\mathbf{x}}}$$

and

$$\bar{h} = \frac{1}{1+e^{-\vec{w} \cdot \tilde{\mathbf{x}}}} = h[\vec{w}](\vec{x})$$

If we model log odds using a linear function, the probability is obtained by applying the logistic sigmoid on the result of the linear function.

# Logistic Regression

- Given a set $D$ of training samples:

$$D = \{(\vec{x}_1, c_1), (\vec{x}_2, c_2), \ldots, (\vec{x}_p, c_p)\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $c_k \in \{0, 1\}$.

Recall that $h[\vec{w}](\vec{x}_k) = 1 \,/\, \left(1 + e^{-\vec{w} \cdot \tilde{\mathbf{x}}_k}\right)$ where $\tilde{\mathbf{x}}_k = (x_{k0}, x_{k1} \ldots, x_{kn})$, here $x_{k0} = 1$

**Our goal:** Find $\vec{w}$ such that for every $k = 1, \ldots, p$ we have that $h[\vec{w}](\vec{x}_k) \approx c_k$

- **Binary Cross-entropy:**

$$E(\vec{w}) = -\sum_{k=1}^{p} c_k \log(h[\vec{w}](\vec{x}_k)) + (1 - c_k) \log(1 - h[\vec{w}](\vec{x}_k))$$

# Gradient of the Error Function

Consider the **gradient** of the error function:

$$\nabla E(\vec{w}) = \left( \frac{\partial E}{\partial w_0}(\vec{w}), \ldots, \frac{\partial E}{\partial w_n}(\vec{w}) \right) = \sum_{k=1}^{p} (h[\vec{w}](\vec{x}_k) - c_k) \cdot \tilde{\mathbf{x}}_k$$

## Fact 1

*If* $\nabla E(\vec{w}) = \vec{0} = (0, \ldots, 0)$, *then* $\vec{w}$ *is a global minimum of* $E$.

This follows from the fact that $E$ is convex.

Using the squared error with the logistic sigmoid would lead to a non-convex error with several minima!

# Logistic Regression – Learning

**Gradient Descent:**

- Weights $\vec{w}^{(0)}$ are initialized randomly close to $\vec{0}$.
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

$$\begin{aligned} \vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \nabla E(\vec{w}^{(t)}) \\ &= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \Big( h[\vec{w}^{(t)}](\vec{x}_k) - c_k \Big) \cdot \tilde{\mathbf{x}}_k \end{aligned}$$

  Here $0 < \varepsilon \leq 1$ is the learning rate.

Note that the algorithm is almost similar to the batch perceptron algorithm!

## Proposition

*For sufficiently small $\varepsilon > 0$, the sequence $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$ converges (in a component-wise manner) to the global minimum of the error function $E$.*

# Logistic Regression - Using the Trained Model

We have already trained our logistic regression model, i.e., we have a vector of weights $\vec{w} = (w_0, w_1, \ldots, w_n)$.

The model is the function $h[\vec{w}]$ which for a given feature vector $\vec{x} = (x_1, \ldots, x_n)$ returns the probability

$$h[\vec{w}](\vec{x}) = \frac{1}{1 + e^{-\left(w_0 + \sum_{k=1}^{n} w_k x_k\right)}}$$

that $\vec{x}$ belongs to the class 1.

To decide whether a given $\vec{x}$ belongs to the class 1 we use $h[\vec{w}]$ as a Bayes classifier: Assign $\vec{x}$ to the class 1 iff $h[\vec{w}](\vec{x}) \geq 1/2$.

Other thresholds can also be used depending on the application and properties of the model. In such a case, given a threshold $\xi \in [0, 1]$, assign $\vec{x}$ to the class 1 iff $h[\vec{w}](\vec{x}) \geq \xi$.

# Maximum Likelihood vs Cross-entropy (Dim 1)

**Fix a training set** $D = \{(x_1, c_1), (x_2, c_2), \ldots, (x_p, c_p)\}$
Generate a sequence $c'_1, \ldots, c'_p \in \{0, 1\}^p$ where each $c'_k$ has been generated independently by the Bernoulli trial generating 1 with probability

$$h[w_0, w_1](x_k) = \frac{1}{1 + e^{-(w_0 + w_1 \cdot x_k)}}$$

and 0 otherwise.

Here $w_0, w_1$ are **unknown weights**.

How "probable" is it to generate the correct classes $c_1, \ldots, c_p$ ?

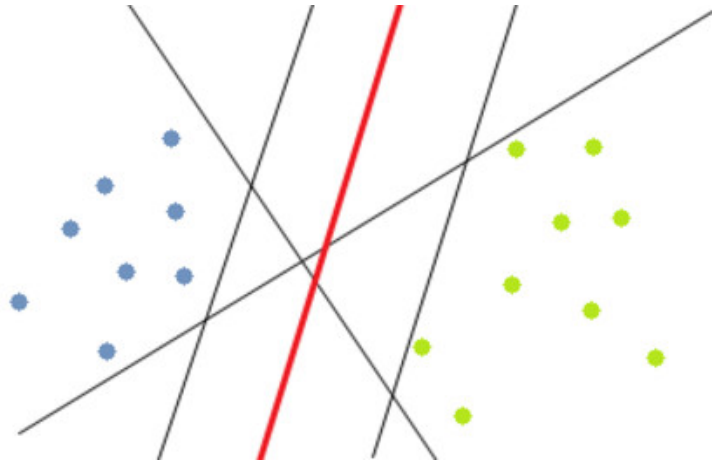
The following conditions are equivalent:

- $w_0, w_1$ minimize the binary cross-entropy $E$
- $w_0, w_1$ maximize the likelihood (i.e., the "probability") of generating the correct values $c_1, \ldots, c_p$ using the above described Bernoulli trials (i.e., that $c'_k = c_k$ for all $k = 1, \ldots, p$)

Note that the above equivalence is a property of the cross-entropy and is not dependent on the "implementation" of $h[w_0, w_1](x_k)$ using the logistic sigmoid.

# Support Vector Machines (SVM)

# SVM Idea – Which Linear Classifier is the Best?



Benefits of maximum margin:

- Intuitively, the maximum margin is good w.r.t. generalization.
- Only the *support vectors* (those on the margin) matter; others can, in principle, be ignored.

# Support Vector Machines (SVM)

Notation:

- $\vec{w} = (w_0, w_1, \ldots, w_n)$ a vector of weights,
- $\underline{\vec{w}} = (w_1, \ldots, w_n)$ a vector of all weights except $w_0$,
- $\vec{x} = (x_1, \ldots, x_n)$ a (generic) feature vector.
- $\tilde{\mathbf{x}} = (x_0, x_1, \ldots, x_n)$ an augmented feature vector where $x_0 = 1$.

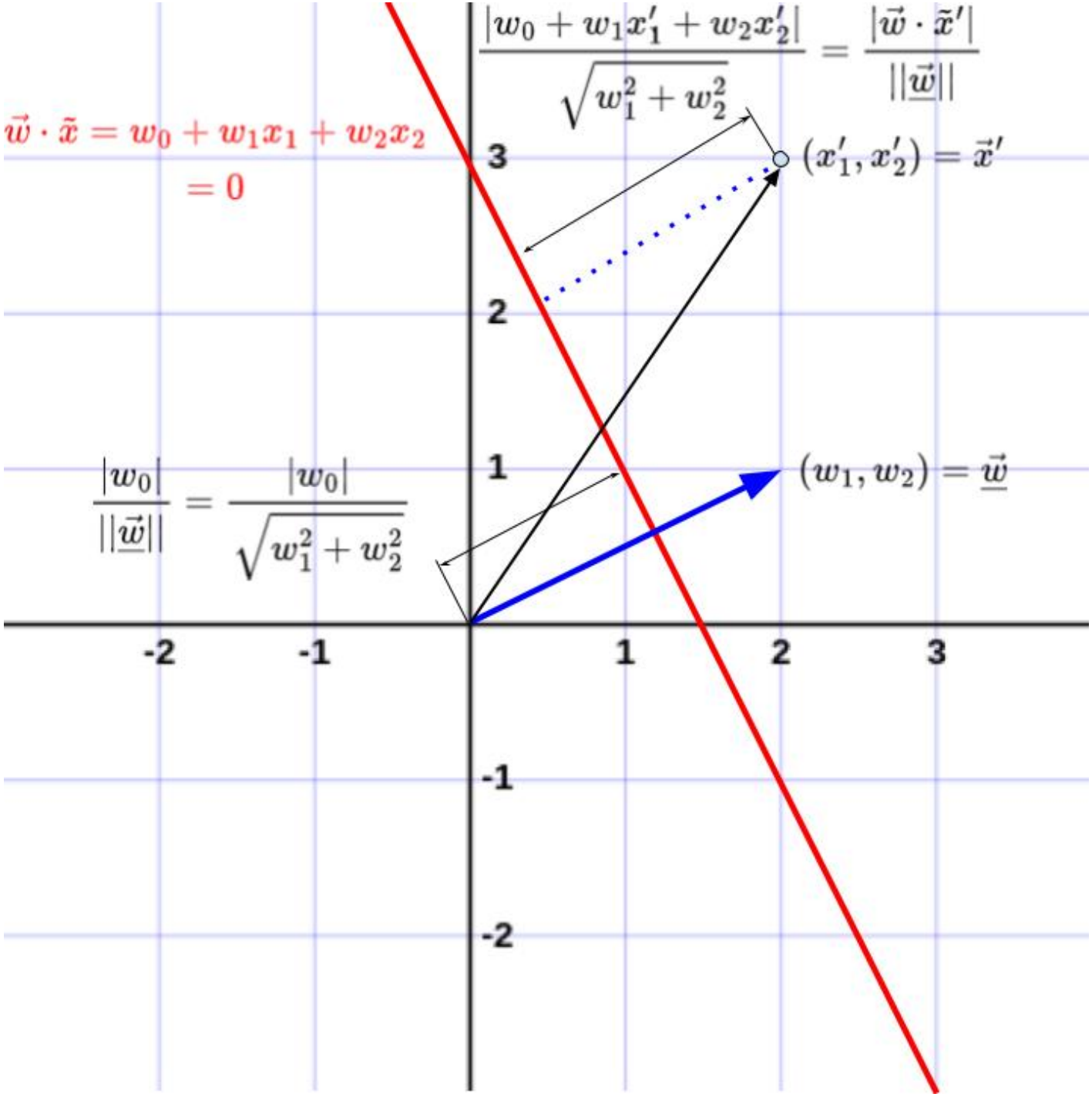
Consider a linear classifier:

$$h[\vec{w}](\vec{x}) := \begin{cases} 1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i = \vec{w} \cdot \tilde{\mathbf{x}} \geq 0 \\ -1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i = \vec{w} \cdot \tilde{\mathbf{x}} < 0 \end{cases}$$

The *distance* of $\vec{x}$ from the separating hyperplane determined by $\vec{w}$ is

$$d[\vec{w}](\vec{x}) = \frac{|\vec{w} \cdot \tilde{\mathbf{x}}|}{\|\underline{\vec{w}}\|}$$

Recall that $\vec{w} \cdot \tilde{\mathbf{x}}$ is positive for $\vec{x}$ on the side to which $\underline{\vec{w}}$ points and negative on the opposite side.


$\frac{|w_0 + w_1 x_1' + w_2 x_2'|}{\sqrt{w_1^2 + w_2^2}} = \frac{|\vec{w} \cdot \vec{x}'|}{||\vec{w}||}$
$\vec{w} \cdot \vec{x} = w_0 + w_1 x_1 + w_2 x_2$
$= 0$
$(x_1', x_2') = \vec{x}'$
$(w_1, w_2) = \vec{w}$
$\frac{|w_0|}{||\vec{w}||} = \frac{|w_0|}{\sqrt{w_1^2 + w_2^2}}$
3
2
1
-1
-2
-2
-1
1
2
3

# Margin

- Given a training set

$$D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_p, y_p)\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in X \subseteq \mathbb{R}^n$ and $y_k \in \{-1, 1\}$.

- Assume that $D$ is linearly separable, let $\vec{w}$ *be consistent with* $D$.

*Margin* of $\vec{w}$ is twice the minimum distance between feature vectors $\vec{x}_k$ and the separating hyperplane determined by $\vec{w}$, i.e.,

$$2 \min_k d[\vec{w}](\vec{x}_k) = 2 \min_k \frac{|\vec{w} \cdot \tilde{\mathbf{x}}_k|}{\|\vec{w}\|}$$

- Our goal is to find $\vec{w}$ consistent with $D$ that maximizes the margin.

Note that to maximize the margin it suffices to maximize $\min_k \frac{|\vec{w} \cdot \tilde{\mathbf{x}}_k|}{\|\vec{w}\|}$ over $\vec{w}$ consistent with $D$.

# Finding the Maximum Margin Classifier

We want to maximize the minimum distance of the feature vectors $\vec{x}_k$ from the separating hyperplane determined by $\vec{w}$.

Formally, we use the following:

To maximize the margin, find $\vec{w}$ *maximizing*

$$\min_k \frac{|\vec{w} \cdot \tilde{\mathbf{x}}_k|}{||\vec{w}||} \qquad (= \text{the distance of closest } \vec{x}_k\text{'s to the sep. hyperplane})$$

over the following constraints

$$\vec{w} \cdot \tilde{\mathbf{x}}_k > 0 \text{ for all } k \text{ satisfying } y_k = 1$$

$$\vec{w} \cdot \tilde{\mathbf{x}}_k < 0 \text{ for all } k \text{ satisfying } y_k = -1$$

(the contraints make sure that $\vec{w}$ is consistent with the training set $D$)

To maximize the margin, find $\vec{w}$ *maximizing*

$$\min_k \frac{|\vec{w} \cdot \tilde{\mathbf{x}}_k|}{||\vec{w}||}$$

over the following constraints

$$\vec{w} \cdot \tilde{\mathbf{x}}_k > 0 \text{ for all } k \text{ satisfying } y_k = 1$$

$$\vec{w} \cdot \tilde{\mathbf{x}}_k < 0 \text{ for all } k \text{ satisfying } y_k = -1$$

Can be made more succinct:

To maximize the margin, find $\vec{w}$ *maximizing*

$$\min_k \frac{y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k}{||\vec{w}||} \quad \text{over} \quad \min_k (y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k) > 0$$

The reason is that $\vec{w}$ is consistent with $D$. That is, $\vec{w} \cdot \tilde{x}_k > 0$ for $y_k = 1$, and $\vec{w} \cdot \tilde{x}_k < 0$ for $y_k = -1$.

To maximize the margin, find $\vec{w}$ *maximizing*

$$\min_k \frac{y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k}{\|\vec{w}\|} \quad \text{over} \quad \min_k (y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k) > 0$$

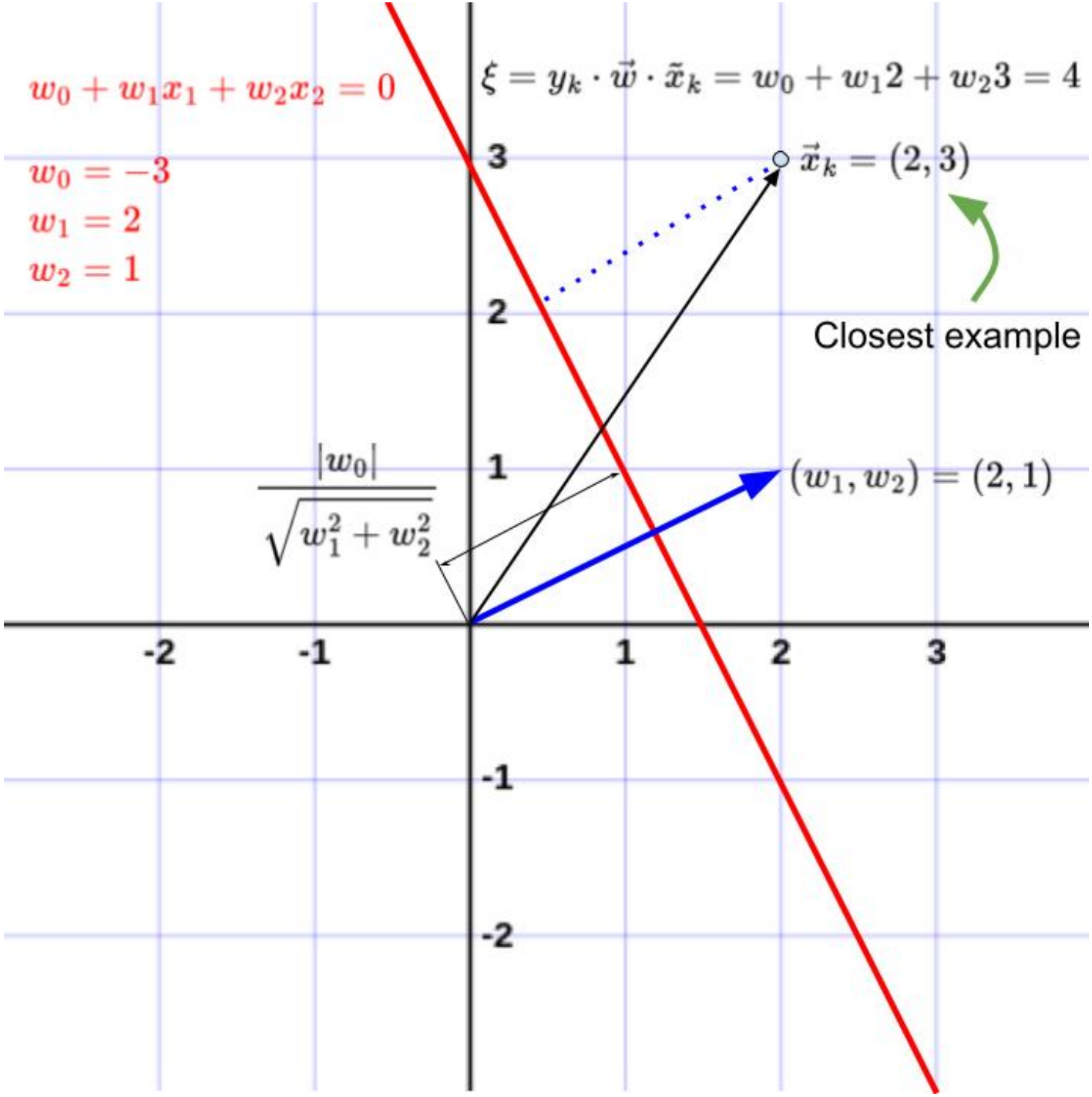
**Observation:** For every $\vec{w}$ satisfying $\min_k (y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k) > 0$ there is $\vec{w}'$ satisfying $\min_k (y_k \cdot \vec{w}' \cdot \tilde{\mathbf{x}}_k) = 1$ such that

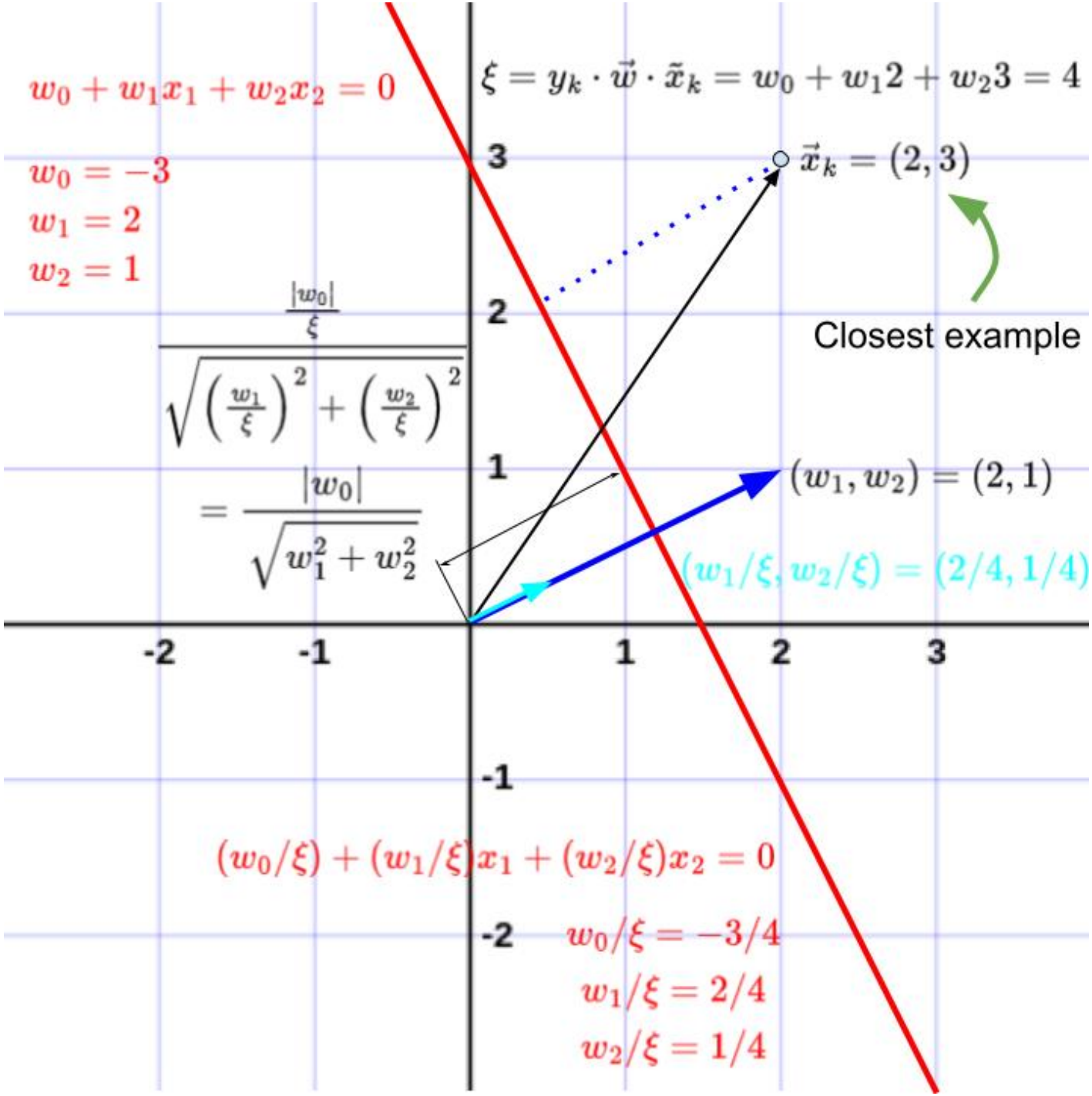
$$\min_k \frac{y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k}{\|\vec{w}\|} = \min_k \frac{y_k \cdot \vec{w}' \cdot \tilde{\mathbf{x}}_k}{\|\vec{w}'\|}$$

**Proof:** Just consider $\vec{w}' = \vec{w}/\xi$ where $\xi = \min_k (y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k)$. □

To maximize the margin, find $\vec{w}$ *maximizing*

$$\min_k \frac{y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k}{\|\vec{w}\|} \quad \text{over} \quad \min_k (y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k) = 1$$


$w_0 + w_1 x_1 + w_2 x_2 = 0$
$w_0 = -3$
$w_1 = 2$
$w_2 = 1$
$\xi = y_k \cdot \vec{w} \cdot \tilde{x}_k = w_0 + w_1 2 + w_2 3 = 4$
$\vec{x}_k = (2, 3)$
Closest example
$\frac{|w_0|}{\sqrt{w_1^2 + w_2^2}}$
$(w_1, w_2) = (2, 1)$
3
2
1
-1
-2
-2
-1
1
2
3


$w_0 + w_1 x_1 + w_2 x_2 = 0$
$w_0 = -3$
$w_1 = 2$
$w_2 = 1$
$\frac{|w_0|}{\xi}$
$\sqrt{\left(\frac{w_1}{\xi}\right)^2 + \left(\frac{w_2}{\xi}\right)^2}$
$= \frac{|w_0|}{\sqrt{w_1^2 + w_2^2}}$
$\xi = y_k \cdot \vec{w} \cdot \tilde{x}_k = w_0 + w_1 2 + w_2 3 = 4$
$\vec{x}_k = (2, 3)$
Closest example
$(w_1, w_2) = (2, 1)$
$(w_1/\xi, w_2/\xi) = (2/4, 1/4)$
$(w_0/\xi) + (w_1/\xi) x_1 + (w_2/\xi) x_2 = 0$
$w_0/\xi = -3/4$
$w_1/\xi = 2/4$
$w_2/\xi = 1/4$
-2
-1
1
2
3
-2
-1
1
2
3

To maximize the margin, find $\vec{w}$ *maximizing*

$$\min_k \frac{y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k}{\|\vec{w}\|} \quad \text{over} \quad \min_k (y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k) = 1$$

can be further simplified to

To maximize the margin, find $\vec{w}$ *maximizing*

$$\frac{1}{\|\vec{w}\|} \quad \text{over} \quad \min_k (y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k) = 1$$

To maximize the margin, find $\vec{w}$ *maximizing*

$$\frac{1}{\|\vec{w}\|} \quad \text{over} \quad \min_k (y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k) = 1$$

Can be adjusted by loosening the constraints:

To maximize the margin, find $\vec{w}$ *maximizing*

$$\frac{1}{\|\vec{w}\|} \quad \text{over} \quad \min_k (y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k) \geq 1$$

If the latter is solved by $\vec{w}'$ with $\min_k (y_k \cdot \vec{w}' \cdot \tilde{\mathbf{x}}_k) > 1$, then

$$\min_k \frac{y_k \cdot \vec{w}' \cdot \tilde{\mathbf{x}}_k}{||\vec{w}'||} > \frac{1}{||\vec{w}'||} \geq \frac{1}{||\vec{w}||} = \frac{\min_k y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k}{||\vec{w}||}$$

For all $\vec{w}$ satisfying $\min_k (y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k) = 1$, which contradicts the fact that the maximum margin is attained by such a $\vec{w}$.

To maximize the margin, find $\vec{w}$ *maximizing*

$$\frac{1}{||\underline{\vec{w}}||} \quad \text{over} \quad \min_k y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k \geq 1$$

Can be turned into

To maximize the margin, find $\vec{w}$ *minimizing*

$$||\underline{\vec{w}}|| \quad \text{over} \quad \min_k y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k \geq 1$$

And, finally,

To maximize the margin, find $\vec{w}$ *minimizing*

$$\underline{\vec{w}} \cdot \underline{\vec{w}} \quad \text{over} \quad y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k \geq 1 \text{ for all } k$$

Indeed, just note that $||\underline{\vec{w}}|| = \sqrt{\underline{\vec{w}} \cdot \underline{\vec{w}}}$.

# SVM – Optimization

Assume a given training set

$$D = \{(\vec{x}_1, y_1)), (\vec{x}_2, y_2), \ldots, (\vec{x}_p, y_p)\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in X \subseteq \mathbb{R}^n$ and $y_k \in \{-1, 1\}$.
(recall $\tilde{x}_k = (x_{k0}, x_{k1}, \ldots, x_{kn})$ where $x_{k0} = 1$)

Margin maximization as a *quadratic optimization problem:*

Find $\vec{w}$ *minimizing*

$$\underline{\vec{w}} \cdot \underline{\vec{w}}$$

under the constraints

$$y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k \geq 1 \text{ for all } k$$

*Support vectors* are vectors $\vec{x}_k$ closest to the *optimal* separating hyperplane, i.e., those satisfying $y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k = 1$ for a minimizing $\vec{w}$.

# Example

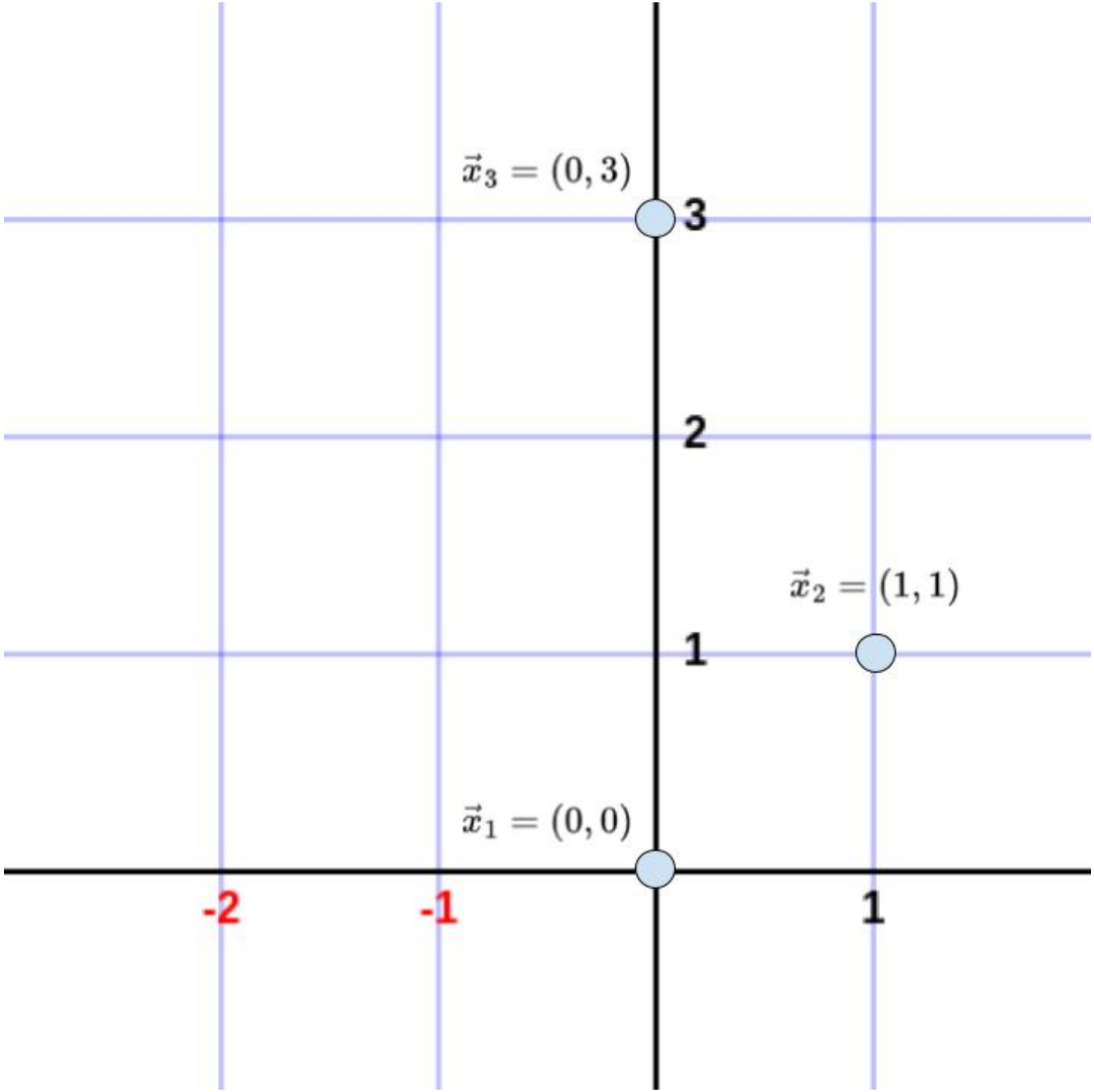
Training set:

$$D = \{((0,0),-1),((1,1),1),((0,3),1)\}$$

That is

$$\begin{aligned} \vec{x}_1 &= (0,0) \\ \vec{x}_2 &= (1,1) \\ \vec{x}_3 &= (0,3) \end{aligned} \qquad \begin{aligned} \tilde{\mathbf{x}}_1 &= (1,0,0) \\ \tilde{\mathbf{x}}_2 &= (1,1,1) \\ \tilde{\mathbf{x}}_3 &= (1,0,3) \end{aligned}$$

$$\begin{aligned} y_1 &= -1 \\ y_2 &= 1 \\ y_3 &= 1 \end{aligned}$$


$\vec{x}_3 = (0, 3)$
3
2
$\vec{x}_2 = (1, 1)$
1
$\vec{x}_1 = (0, 0)$
-2
-1
1

Find $\vec{w}$ minimizing $w_1^2 + w_2^2$ under the constraints

$$\begin{aligned} (-1) \cdot (1w_0 + 0w_1 + 0w_2) = -w_0 &\geq 1 \\ 1 \cdot (1w_0 + 1w_1 + 1w_2) = w_0 + w_1 + w_2 &\geq 1 \\ 1 \cdot (1w_0 + 0w_1 + 3w_2) = w_0 + 3w_2 &\geq 1 \end{aligned}$$

It can be solved using a quadratic programming solver.

To solve by hand, assume that we know that $\vec{x}_1$ and $\vec{x}_2$ are support vectors.

Find $\vec{w}$ minimizing $w_1^2 + w_2^2$ under the constraints

$$\begin{aligned} -w_0 &= 1 \\ w_0 + w_1 + w_2 &= 1 \\ w_0 + 3w_2 &\geq 1 \end{aligned}$$

Note that the equality constraints correspond to our assumption that $\vec{x}_1$ and $\vec{x}_2$ are support vectors.

Find $\vec{w}$ minimizing $w_1^2 + w_2^2$ under the constraints

$$\begin{aligned} -w_0 &= 1 \\ w_0 + w_1 + w_2 &= 1 \\ w_0 + 3w_2 &\geq 1 \end{aligned}$$

Can be transformed to

Find $\vec{w}$ minimizing $w_1^2 + w_2^2$ under the constraints

$$\begin{aligned} w_1 + w_2 &= 2 \\ 3w_2 &\geq 2 \end{aligned}$$

Find $\vec{w}$ minimizing $w_1^2 + w_2^2$ under the constraints

$$w_1 + w_2 = 2$$
$$3w_2 \geq 2$$

Substituting $w_2 = 2 - w_1$ into the quadratic function we obtain

$$w_1^2 + (2 - w_1)^2 = w_1^2 + w_1^2 - 4w_1 + 4 = 2w_1^2 - 4w_1 + 4$$

substituting $w_2 = 2 - w_1$ into the inequality $3w_2 \geq 2$ we obtain

$$6 - 3w_1 \geq 2$$

This reduces our problem to

Find $\vec{w}$ minimizing $2w_1^2 - 4w_1 + 4$ under the constraint $w_1 \leq \frac{4}{3}$

Find $\vec{w}$ minimizing $2w_1^2 - 4w_1 + 4$ under the constraint $w_1 \leq \frac{4}{3}$

Is solved by

$$w_1 = 1$$

From $w_2 = 2 - w_1$ we obtain

$$w_2 = 2 - 1 = 1$$

From $-w_0 = 1$ we obtain

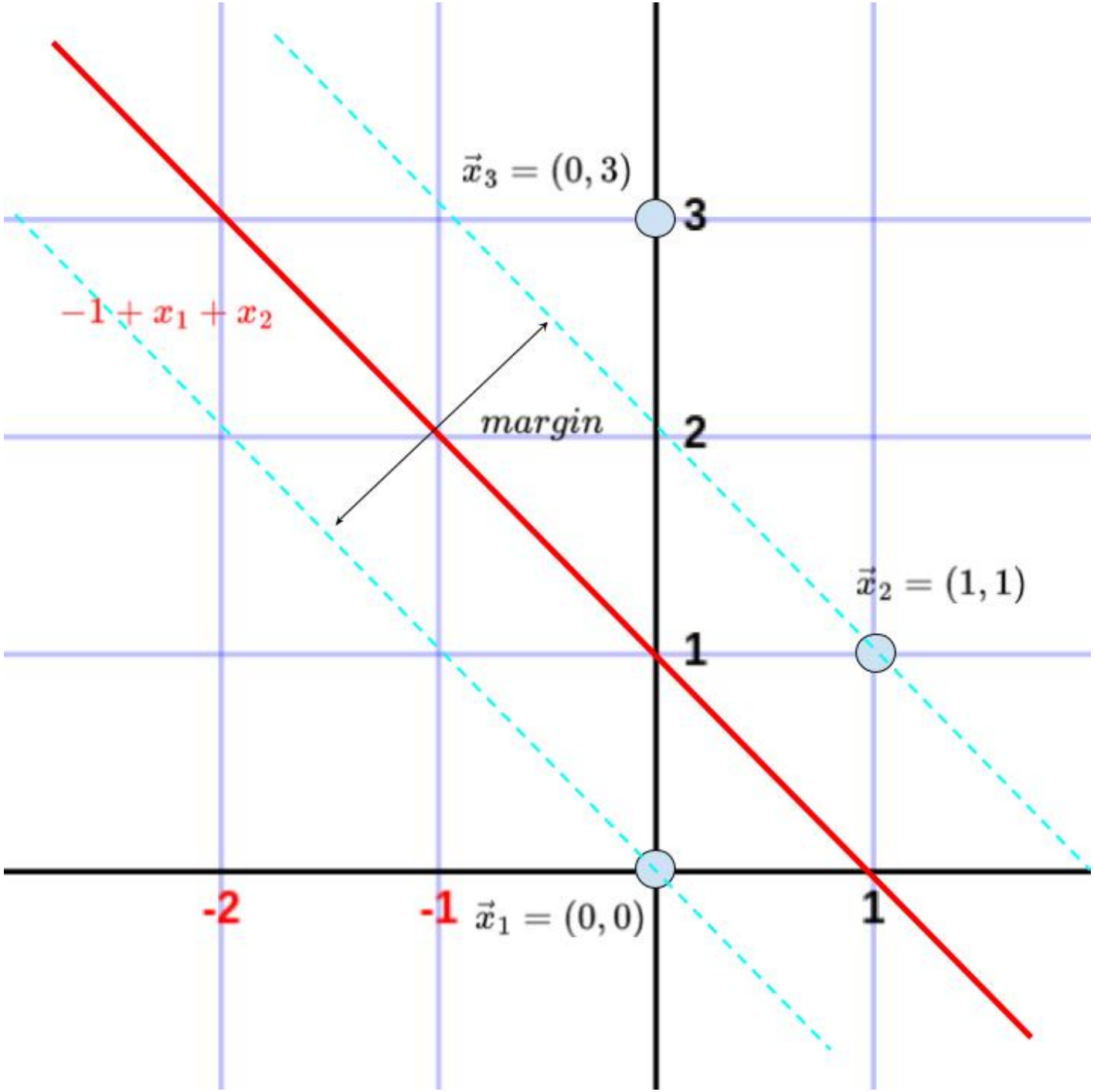
$$w_0 = -1$$

The final model is

$$h[\vec{w}](\vec{x}) = -1 + x_1 + x_2$$

The separating hyperplane is determined by

$$-1 + x_1 + x_2 = 0$$


$\vec{x}_3 = (0, 3)$
3
$-1 + x_1 + x_2$
$margin$
2
$\vec{x}_2 = (1, 1)$
1
-2
-1
$\vec{x}_1 = (0, 0)$
1

# SVM – Optimization

- Need to optimize a quadratic function subject to linear constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems for which efficient methods (and tools) exist.

But why has the SVM been so successful?

... the improvement by finding the maximum margin classifier does not seem to be so strong ... right?

The answer lies in their ability to deal with non-linearly separable sets efficiently using the *kernel trick* (see a later lecture).

# Comments on Algorithms

- The main bottleneck of SVM is quadratic programming (QP) complexity. A naive QP solver has cubic complexity.
- For small problems, any general-purpose optimization algorithm can be used.
- For large problems, this is usually not possible; many methods avoiding direct solutions have been devised.
- These methods usually decompose the optimization problem into a sequence of smaller ones. Intuitively,
  - starts with a (smaller) subset of training examples.
  - Find an optimal solution using any solver.
  - Afterwards, only support vectors matter in the solution! Leave only them in the training set, and add new training examples.
  - This iterative procedure decreases the (general) cost function.

# Soft-margin SVM

Trade-off few misclassifications with a wide margin for the rest.

Find $\vec{w}$ *minimizing*

$$\underline{\vec{w}} \cdot \underline{\vec{w}} + C \sum_k \zeta_k \qquad C \text{ is a hyperparameter}$$

under the constraints

$$y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k \geq 1 - \zeta_k \text{ for all } k$$

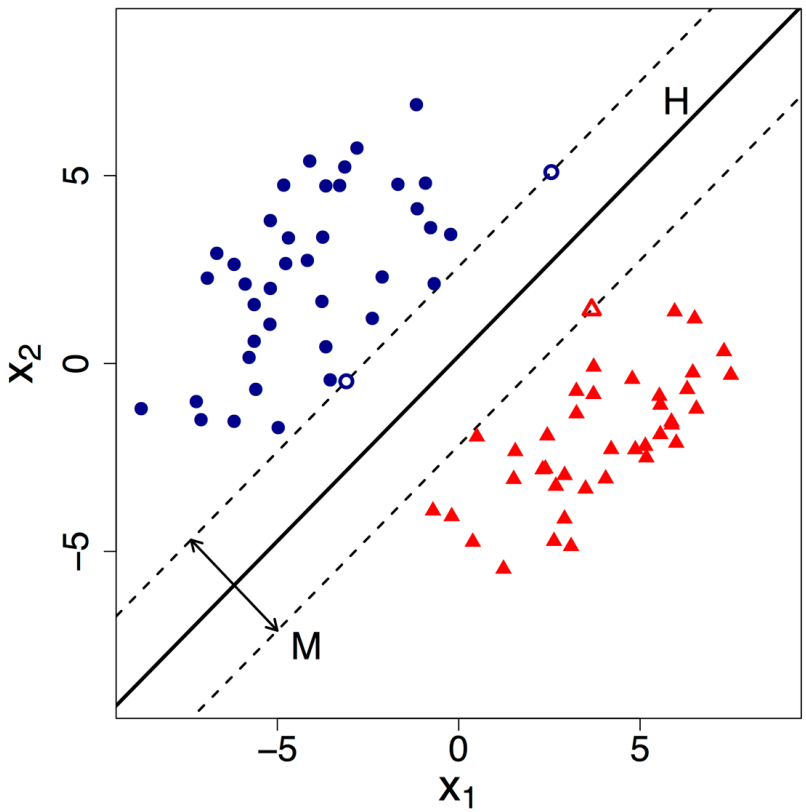
$$\zeta_k \geq 0 \text{ for all } k$$

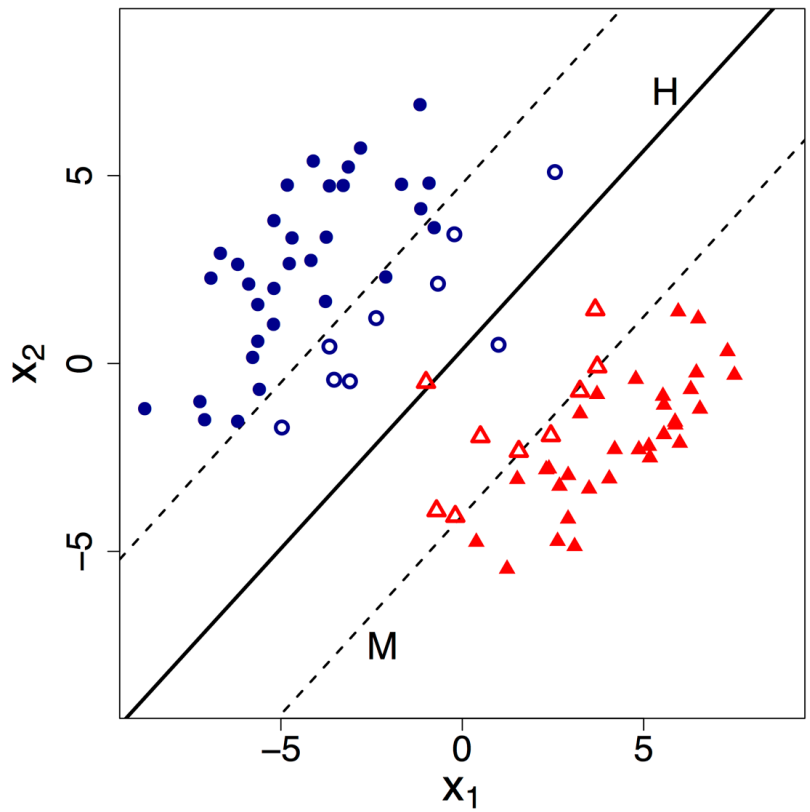
Which is the same as the following *unconstrained* optimization:

Find $\vec{w}$ *minimizing* the *hinge loss*

$$\underline{\vec{w}} \cdot \underline{\vec{w}} + C \sum_k \max(0, 1 - y_k \cdot \vec{w} \cdot \tilde{\mathbf{x}}_k)$$

# Hard vs Soft Margin SVM




Source: Dishaa Agarwal

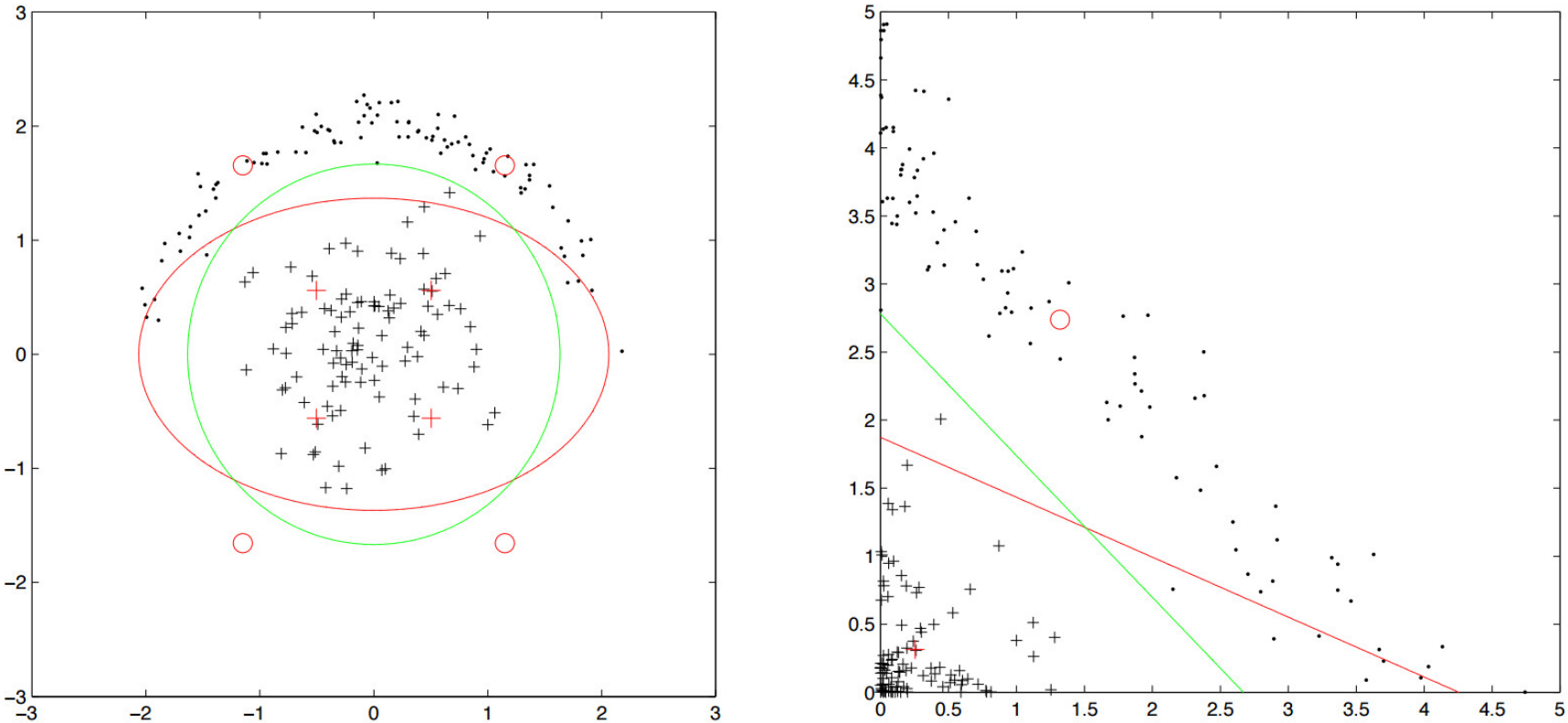
https://www.analyticsvidhya.com/blog/2021/04/insight-into-svm-support-vector-machine-along-with-code/

# Comments on SVM

- SVMs were originally proposed by Boser, Guyon, and Vapnik in 1992, and gained increasing popularity in the late 1990s.
- SVMs are currently among the best performers for several classification tasks ranging from text to genomic data.
- SVMs can be applied to complex data types beyond feature vectors (e.g., graphs, sequences, relational data) by designing kernel functions for such data.
- SVM techniques have been extended to several tasks, such as regression [Vapnik et al. '97], principal component analysis [Schölkopf et al. '99], etc.

# Kernel Methods

# Quadratic Decision Boundary



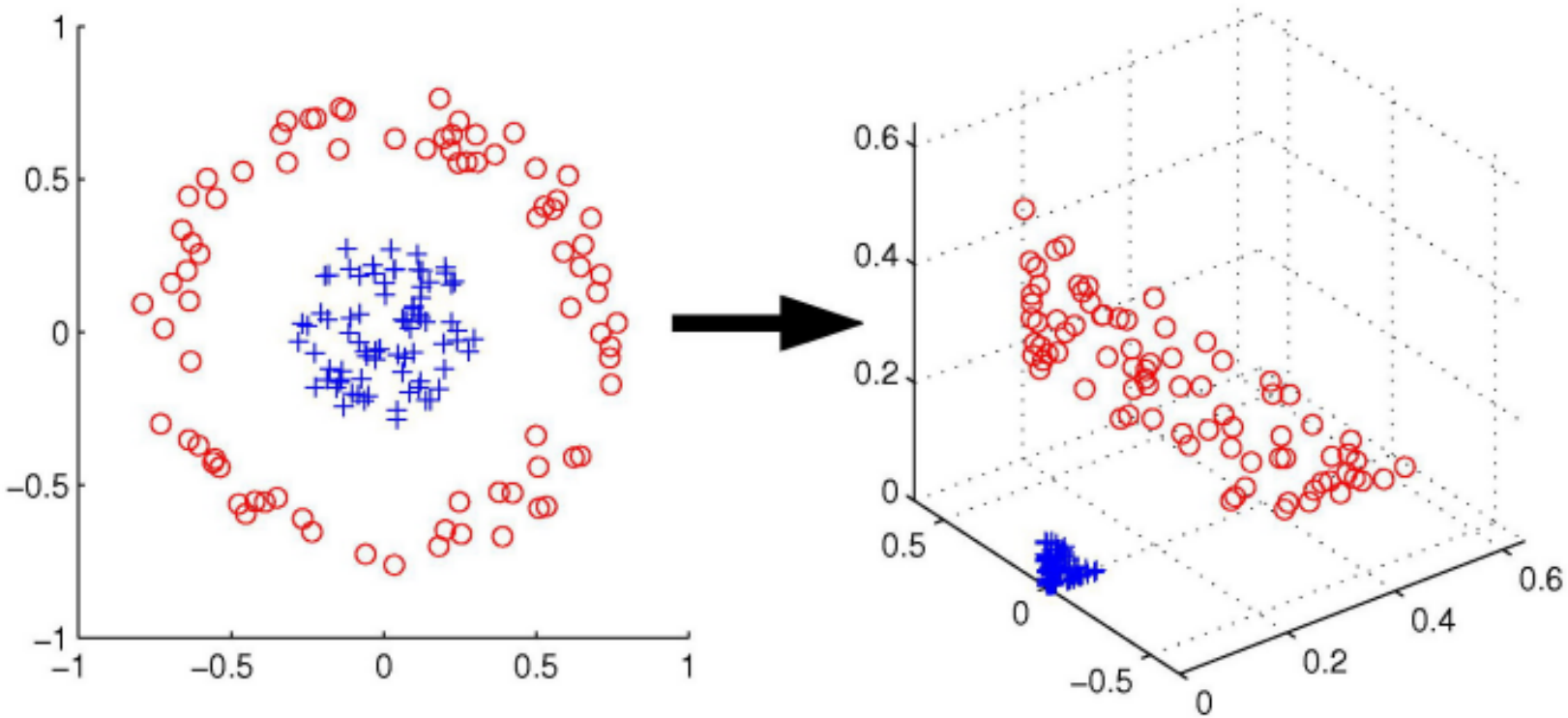

Left: The original set, Right: Transformed using the square of features.

Right: the green line is a separating hyperplane in transformed space.

Left: the green ellipse maps exactly to the green line.

How to classify (in the original space): First, transform a given feature vector by squaring the features, then use a linear classifier.

# Another Solution




Mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ so that there is "more space" for linear separation.

# Do We Need to Map Explicitly?

In general, mapping to (much) higher feature space helps (there are more "degrees of freedom" so linear separability might get a chance).

However, the complexity of learning grows (quickly) with dimension.

Sometimes it's even beneficial to map to infinite-dimensional spaces.

To avoid explicit construction of the higher dimensional feature space, we use the so-called *kernel trick*.

But first, we need to *dualize* our learning algorithm.

# Linear Regression

- Given a set $D$ of training examples:

$$D = \{(\vec{x}_1, f_1), (\vec{x}_2, f_2), \ldots, (\vec{x}_p, f_p)\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $f_k \in \mathbb{R}$.

- **Our goal:** Find $\vec{w}$ so that $h[\vec{w}](\vec{x_k}) = \vec{w} \cdot \tilde{\mathbf{x}}_k$ is close to $f_k$ for every $k = 1, \ldots, p$.

Recall that $\tilde{\mathbf{x}}_k = (x_{k0}, x_{k1} \ldots, x_{kn})$ where $x_{k0} = 1$.

- **Squared Error Function:**

$$E(\vec{w}) = \frac{1}{2}\sum_{k=1}^{p}(\vec{w} \cdot \tilde{\mathbf{x}}_k - f_k)^2 = \frac{1}{2}\sum_{k=1}^{p}\left(\sum_{i=0}^{n} w_i x_{ki} - f_k\right)^2$$

# Regularized Linear Regression

**Regularized Squared Error Function:**

$$E(\vec{w}) \;=\; \frac{1}{2}\sum_{k=1}^{p}(\vec{w}\cdot\tilde{\mathbf{x}}_k - f_k)^2 + \vec{w}\cdot\vec{w}$$

Intuition: the added term $\vec{w}\cdot\vec{w}$ prevents growth of weights.

**The Representer Theorem:** The weight vector $\vec{w}^*$ minimizing the regularized squared error function can be written as

$$\vec{w}^* = \sum_{i=1}^{p}\alpha_i f_i\,\tilde{\mathbf{x}}_i \qquad \text{Here } \alpha_1,\ldots,\alpha_p \text{ are suitable coefficients.}$$

Substituting this expression for weights in $E$ gives

$$E'(\vec{w}^*) = \frac{1}{2}\sum_{k=1}^{p}\left(\sum_{i=1}^{p}\alpha_i f_i\,\tilde{\mathbf{x}}_i\cdot\tilde{\mathbf{x}}_k - f_k\right)^2 + \sum_{i=1}^{p}\sum_{j=1}^{p}\alpha_i\alpha_j f_i f_j\,\tilde{\mathbf{x}}_i\cdot\tilde{\mathbf{x}}_j$$

and we minimize $E'$ w.r.t. $\alpha_1,\ldots,\alpha_p$. What is this good for??

Given a set $D$ of training examples:

$$D = \{(\vec{x}_1, f_1), (\vec{x}_2, f_2), \ldots, (\vec{x}_p, f_p)\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $f_k \in \mathbb{R}$.

Find $\alpha_1, \ldots, \alpha_p$ *minimizing dual regularized squared error*

$$E'(\vec{w}^*) = \frac{1}{2} \sum_{k=1}^{p} \left( \sum_{i=1}^{p} \alpha_i f_i \, \tilde{\mathbf{x}}_i \cdot \tilde{\mathbf{x}}_k - f_k \right)^2 + \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j f_i f_j \, \tilde{\mathbf{x}}_i \cdot \tilde{\mathbf{x}}_j$$

The resulting coefficients $\alpha_1, \ldots, \alpha_p$ give a weight vector

$$\vec{w}^* = \sum_{i=1}^{p} \alpha_i f_i \, \tilde{\mathbf{x}}_i$$

which in turn gives a linear model

$$h[\vec{w}^*](\vec{x}) = \vec{w}^* \tilde{\mathbf{x}} = \sum_{i=1}^{p} \alpha_i f_i \, \tilde{\mathbf{x}}_i \cdot \tilde{\mathbf{x}}$$

Note that all $\tilde{\mathbf{x}}, \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j, \tilde{\mathbf{x}}_k$ occur in dot products with themselves!

Find $\vec{\alpha} = (\alpha_1, \ldots, \alpha_p)$ *minimizing dual regularized squared error*

$$E'(\vec{w}^*) = \frac{1}{2}\sum_{k=1}^{p}\left(\sum_{i=1}^{p}\alpha_i f_i\, \tilde{\mathbf{x}}_i \cdot \tilde{\mathbf{x}}_k - f_k\right)^2 + \sum_{i=1}^{p}\sum_{j=1}^{p}\alpha_i\alpha_j f_i f_j\, \tilde{\mathbf{x}}_i \cdot \tilde{\mathbf{x}}_j$$

Linear model: $h[\vec{\alpha}](\vec{x}) = \sum_{i=1}^{p}\alpha_i f_i\, \tilde{\mathbf{x}}_i \cdot \tilde{\mathbf{x}}$

Do we need to use the dot product in the above procedure? NO!

Find $\vec{\alpha} = (\alpha_1, \ldots, \alpha_p)$ *minimizing kernel dual regularized squared error*

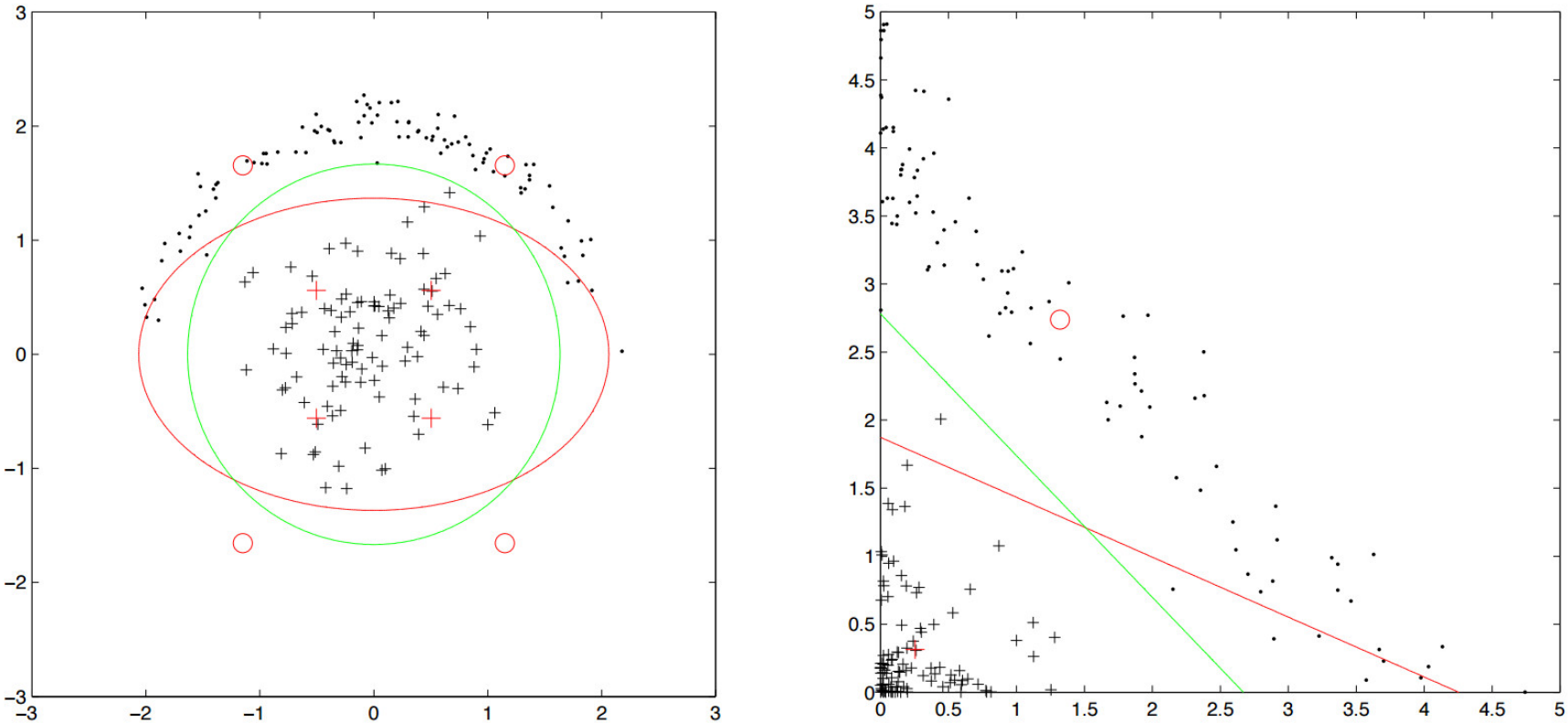
$$E'(\vec{w}^*) = \frac{1}{2}\sum_{k=1}^{p}\left(\sum_{i=1}^{p}\alpha_i f_i\, \kappa(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_k)) - f_k\right)^2 + \sum_{i=1}^{p}\sum_{j=1}^{p}\alpha_i\alpha_j f_i f_j\, \kappa(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$$

Non-linear model: $h[\vec{\alpha}](\vec{x}) = \sum_{i=1}^{p}\alpha_i f_i\, \kappa(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}})$

Here $\kappa$ is a **kernel function**. But now what is the trick?

The trick is that suitable kernel functions $\kappa$ correspond to dot products in transformed spaces!

# Recall the Quadratic Decision Boundary




Left: The original set, Right: Transformed using the square of features.

Right: the green line is a separating hyperplane in the transformed space.

Left: the green ellipse maps exactly to the green line.

How to classify (in the original space): Transform a given feature vector by squaring the features, then use a linear classifier.

# Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Consider two instances $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$ and $\tilde{\mathbf{x}}_\ell = (1, x_{\ell 1}, x_{\ell 2})$. Then the scalar product of their corresponding instances $(1, x_{k1}^2, x_{k2}^2)$ and $(1, x_{\ell 1}^2, x_{\ell 2}^2)$, resp., in the quadratic feature space is

$$1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2$$

which resembles (but is not equal to)

$$(\tilde{\mathbf{x}}_k \cdot \tilde{\mathbf{x}}_\ell)^2 = (1 + x_{k1}x_{\ell 1} + x_{k2}x_{\ell 2})^2 =$$
$$= 1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2 + 2x_{k1}x_{\ell 1}x_{k2}x_{\ell 2} + 2x_{k1}x_{\ell 1} + 2x_{k2}x_{\ell 2}$$

But now consider a mapping $\phi$ to $\mathbb{R}^6$ defined by

$$\phi(\tilde{\mathbf{x}}_k) = (1, x_{k1}^2, x_{k2}^2, \sqrt{2}x_{k1}x_{k2}, \sqrt{2}x_{k1}, \sqrt{2}x_{k2})$$

Then $\phi(\tilde{\mathbf{x}}_k) \cdot \phi(\tilde{\mathbf{x}}_\ell) = (\tilde{\mathbf{x}}_k \cdot \tilde{\mathbf{x}}_\ell)^2$

**THE Idea:** Using the kernel $\kappa(\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_\ell) = (\tilde{\mathbf{x}}_k \cdot \tilde{\mathbf{x}}_\ell)^2$ in the kernel, dual regularized squared error corresponds to using the regularized squared error after the transformation $\phi$.

# Quadratic Decision Boundary

Given a set $D$ of training examples:

$$D = \{(\vec{x}_1, f_1), (\vec{x}_2, f_2), \ldots, (\vec{x}_p, f_p)\}$$

Assume that $f_i \in \{1, -1\}$ indicates the class of $\vec{x}_i$.

Yes, I know that squared error regression should not be used for classification!

Considering $\kappa(\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_\ell) = (\tilde{\mathbf{x}}_k \cdot \tilde{\mathbf{x}}_\ell)^2$ in our kernel dual regularized squared error we obtain

Find $\vec{\alpha} = \alpha_1, \ldots, \alpha_p$ *minimizing*

$$E'(\vec{w}) = \frac{1}{2} \sum_{k=1}^{p} \left( \sum_{i=1}^{p} \alpha_i f_i \, (\tilde{\mathbf{x}}_i \cdot \tilde{\mathbf{x}}_k)^2 - f_k \right)^2 + \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j f_i f_j \, (\tilde{\mathbf{x}}_i \cdot \tilde{\mathbf{x}}_j)^2$$

Non-linear classifier: $h[\vec{\alpha}](\vec{x}) = \sum_{i=1}^{p} \alpha_i f_i \, (\tilde{\mathbf{x}}_i \cdot \tilde{\mathbf{x}})^2$

Intuitively, minimizing $E'$ in $\mathbb{R}^2$ gives a separating hyperplane for the input vectors transformed into $\mathbb{R}^5$. This means, that in $\mathbb{R}^2$ it searches for a quadratic (i.e., non-linear) boundary.

# Examples of Kernels

- Linear: $\kappa(\tilde{\mathbf{x}}_\ell, \tilde{\mathbf{x}}_\mathbf{k}) = \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_\mathbf{k}$

  The corresponding mapping $\phi(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}$ is identity (no transformation).

- Polynomial of power $m$: $\kappa(\tilde{\mathbf{x}}_\ell, \tilde{\mathbf{x}}_\mathbf{k}) = (\tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_\mathbf{k})^m$

  The corresponding mapping assigns to $\tilde{\mathbf{x}} \in \mathbb{R}^{n+1}$ the vector $\phi(\tilde{\mathbf{x}})$ in $\mathbb{R}^{\binom{n+m}{m}+1}$.

- Gaussian (radial-basis function): $\kappa(\tilde{\mathbf{x}}_\ell, \tilde{\mathbf{x}}_\mathbf{k}) = e^{-\frac{\|\tilde{\mathbf{x}}_\ell - \tilde{\mathbf{x}}_\mathbf{k}\|^2}{2\sigma^2}}$

  The corresponding mapping $\phi$ maps $\tilde{\mathbf{x}}$ to an *infinite-dimensional* vector $\phi(\tilde{\mathbf{x}})$ which is, in fact, a Gaussian function; a combination of such functions for support vectors is then the separating hypersurface.

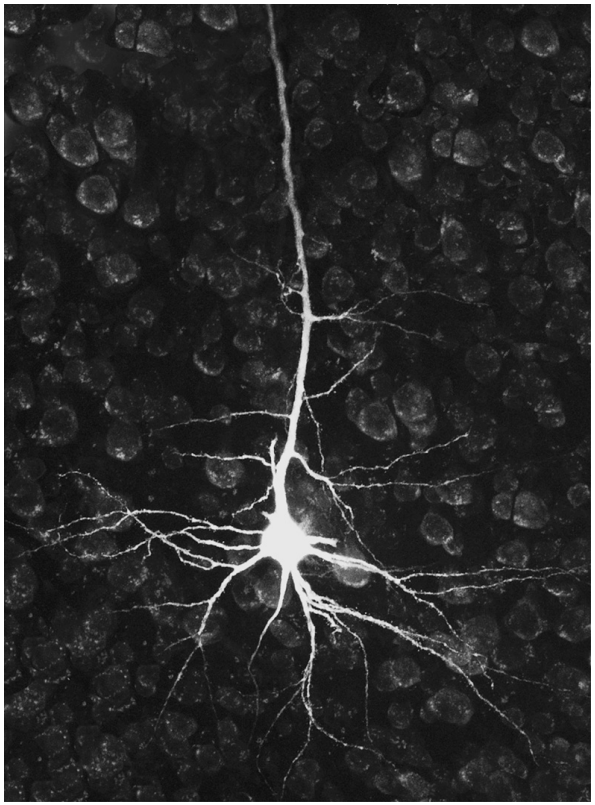
- ...

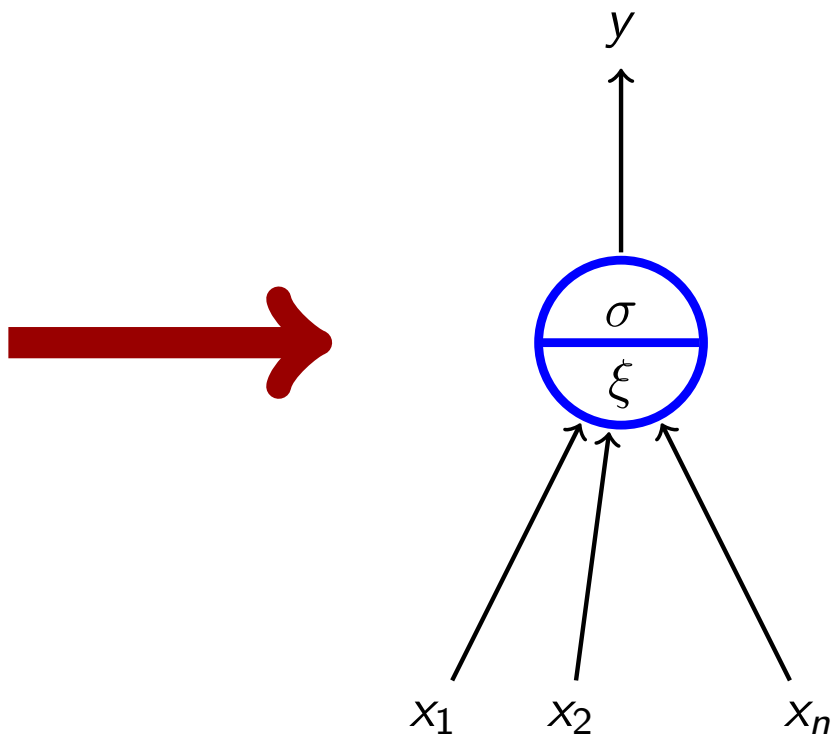
Choosing kernels remains to be the black magic of kernel methods. They are usually chosen based on trial and error (of course, experience and additional insight into data help).

A similar trick can be done with (soft-margin) support vector machines.

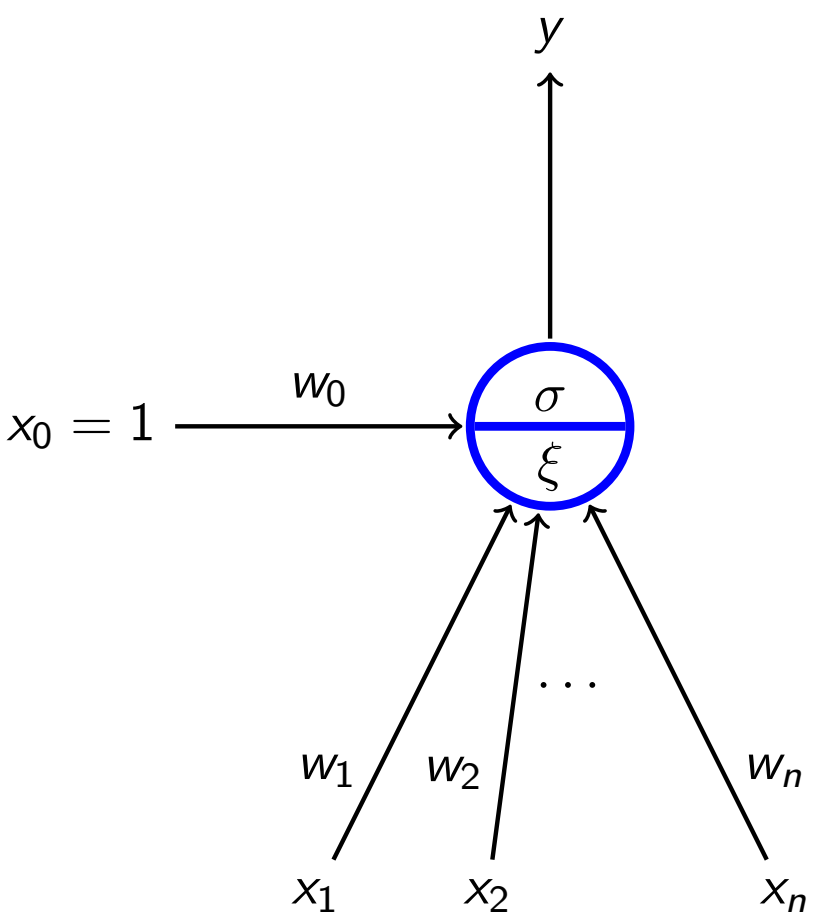
# Neural Networks

# (Primitive) Mathematical Model of Neuron





# Formal neuron




- $x_1, \ldots, x_n$ real *inputs*
- $x_0$ special input, always 1
- $w_0, w_1, \ldots, w_n$ real *weights*
- $\xi = w_0 + \sum_{i=1}^{n} w_i x_i$ *inner potential*;

  In general, other potentials are considered (e.g. Gaussian), more on this in PV021.
- $y$ *output* defined by $y = \sigma(\xi)$ where $\sigma$ is an *activation function*.

  We consider several activation functions.

  e.g., *linear threshold function*

$$\sigma(\xi) = sgn(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

# Formal Neuron vs Linear Models

- If $\sigma$ is a linear threshold function

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

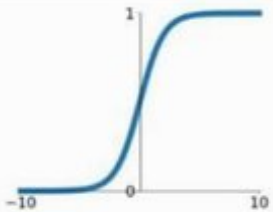
  We obtained a linear classifier.
- If $\sigma$ is identity, i.e., $\sigma(\xi) = \xi$, we obtain a linear (affine) function.
- If $\sigma(\xi) = 1/(1 + e^{-\xi})$ we obtain the logistic regression.

Also, other activation functions are used in neural networks!

# Activation Functions

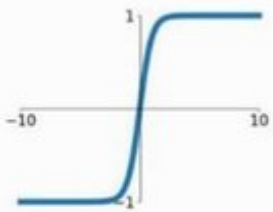
**Sigmoid**

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



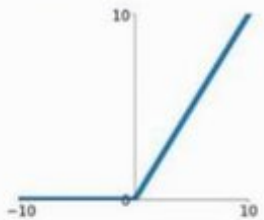

**tanh**

$$\tanh(x)$$



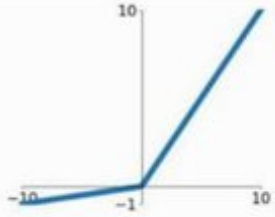

**ReLU**

$$\max(0, x)$$



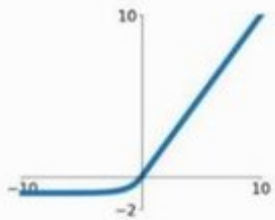

**Leaky ReLU**

$$\max(0.1x, x)$$

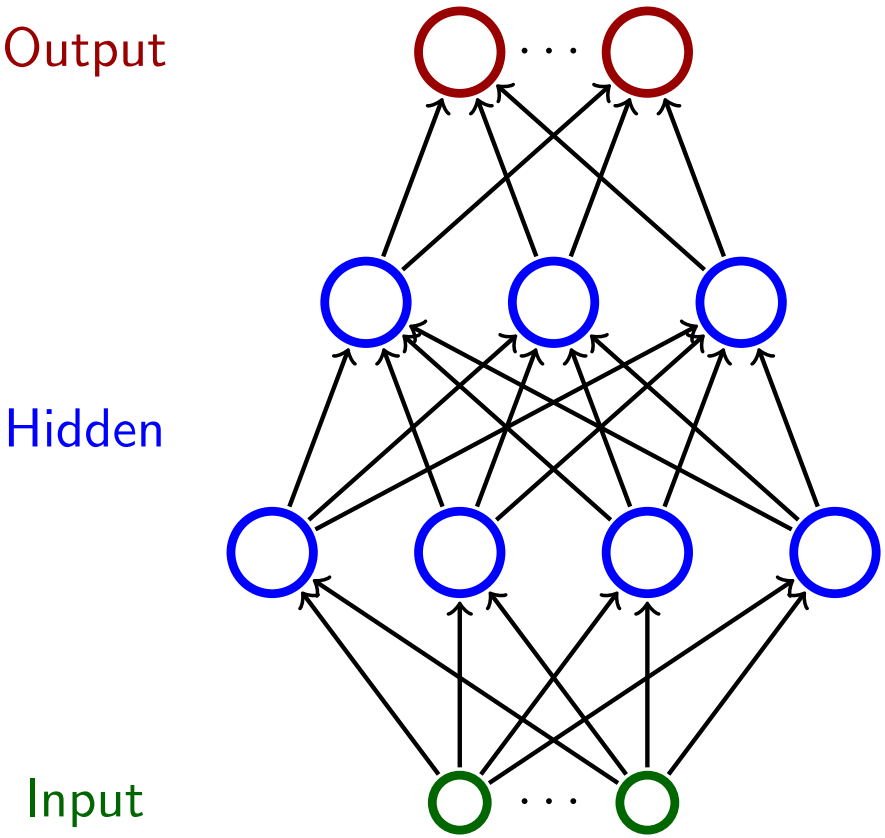



**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# Multilayer Perceptron (MLP)



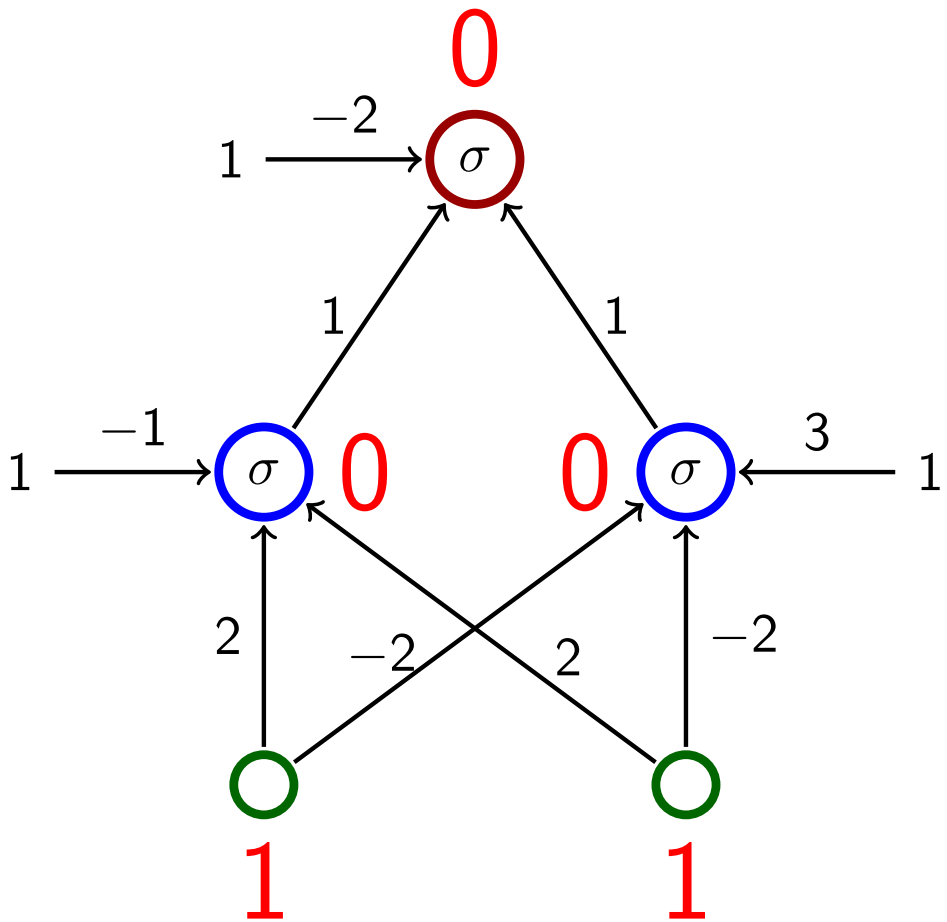

- Neurons are organized in *layers*

  (input layer, output layer, possibly several hidden layers)
- Layers are numbered from 0;

  The input is 0-th
- Neurons in the $\ell$-th layer are connected with all neurons in the $\ell + 1$-th layer

**Intuition:** The network computes a function: Assign input values to the input neurons and 0 to the rest. Proceed upwards through the layers, one layer per step. In the $\ell$-th step consider output values of neurons in $\ell - 1$-th layer as inputs to neurons of the $\ell$-th layer. Compute output values of neurons in the $\ell$-th layer.

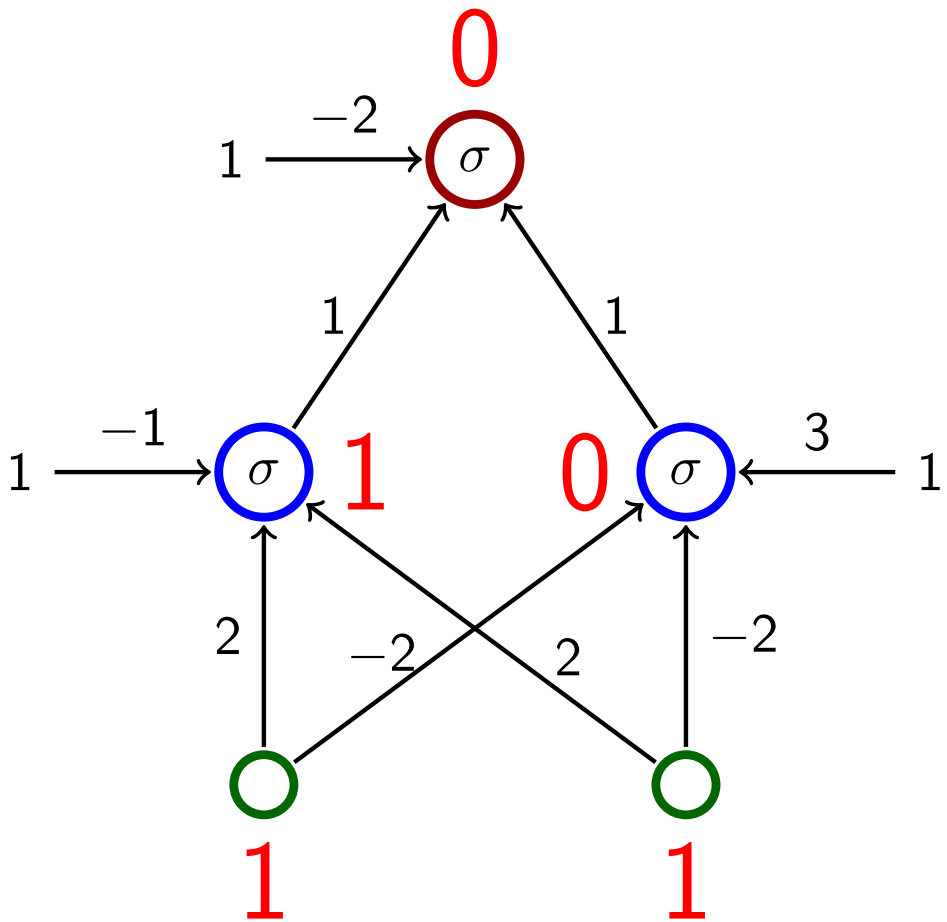
# Example




- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

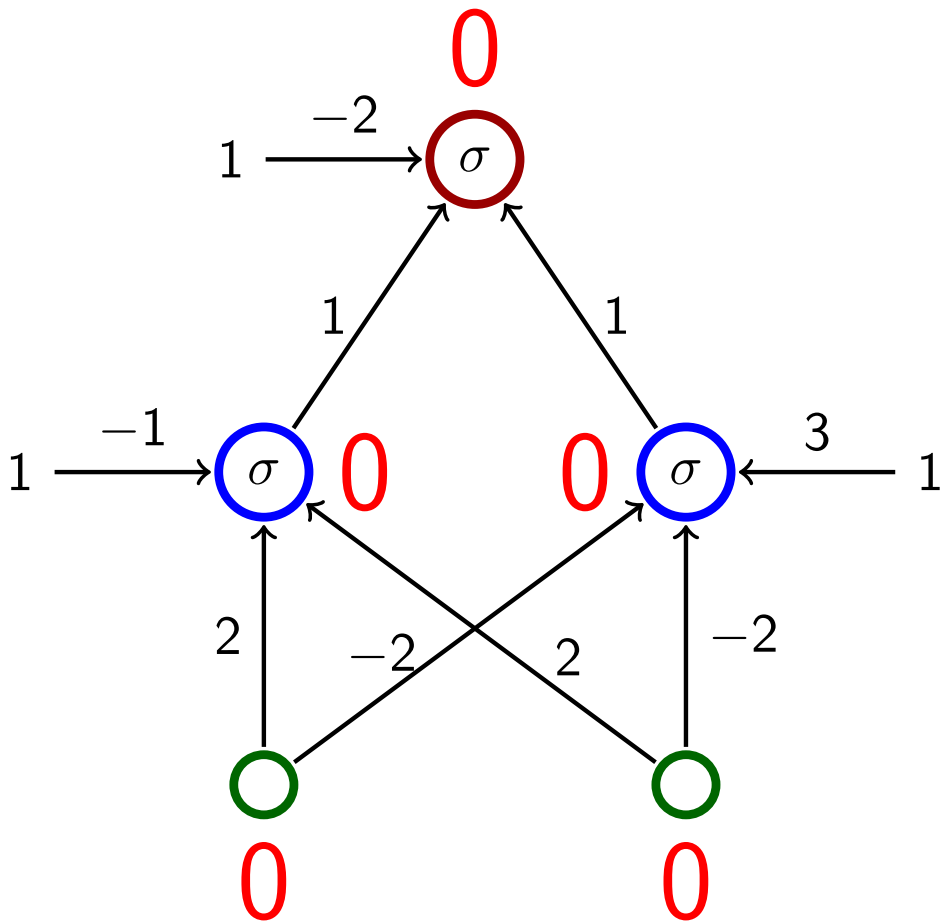
# Example




- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

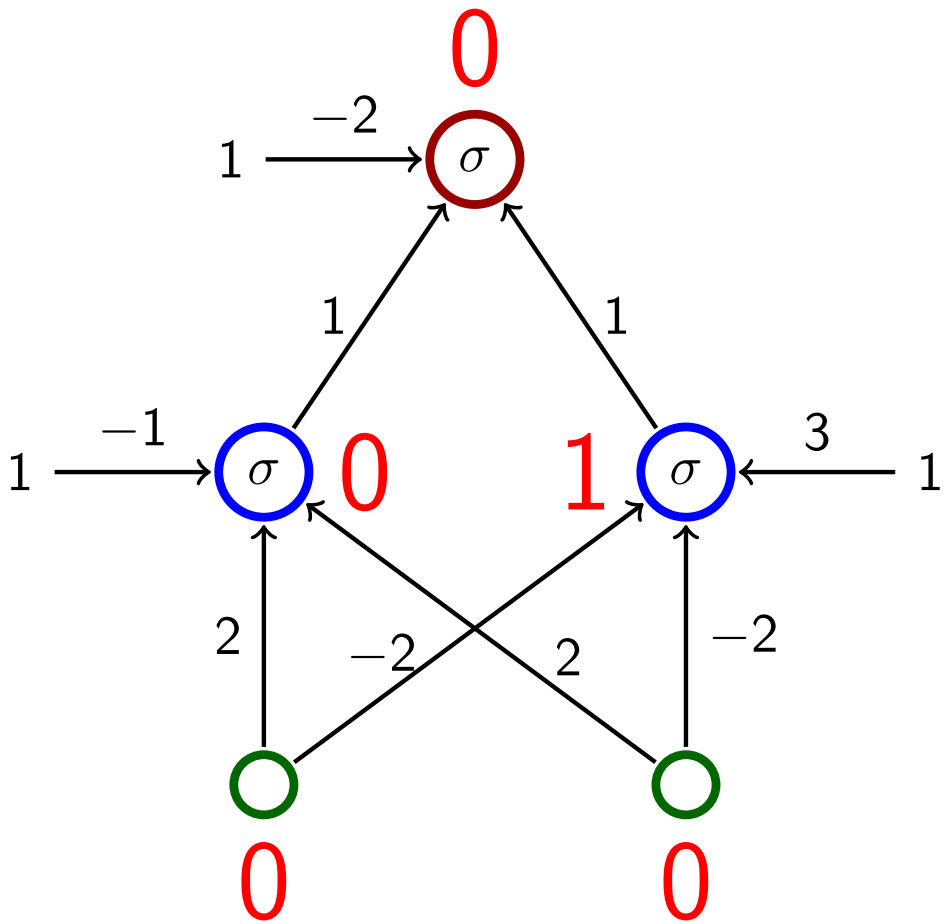
# Example




- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

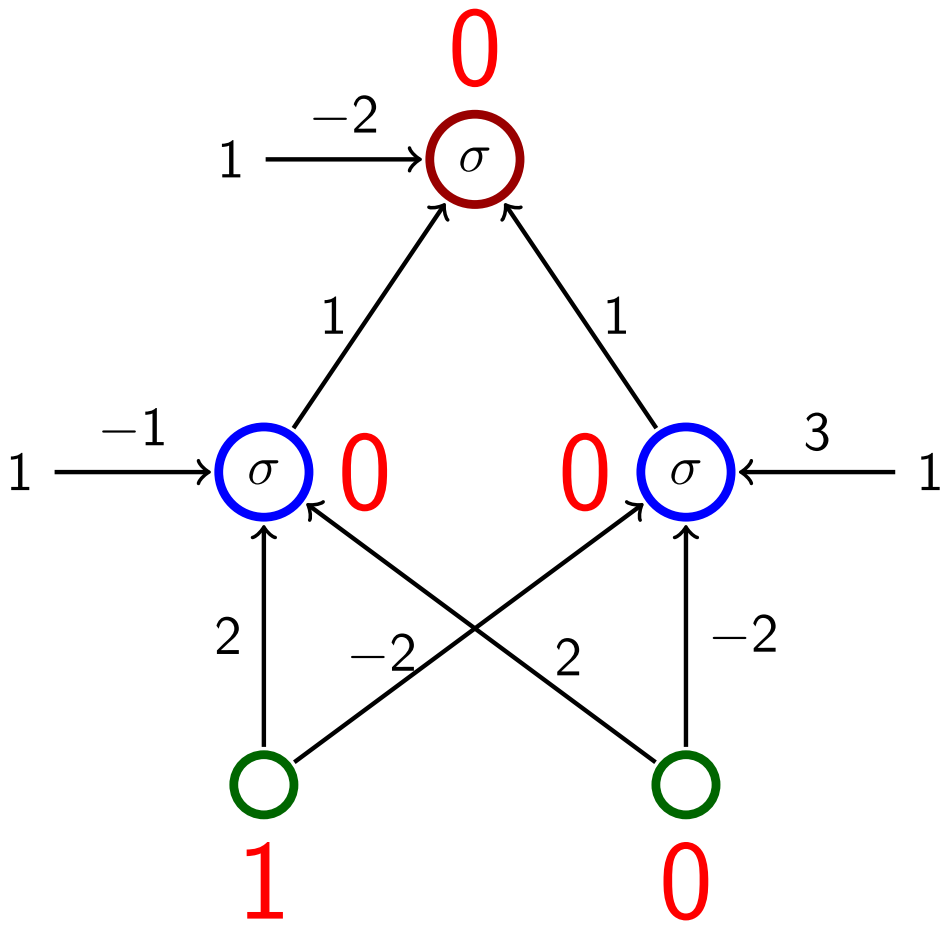
# Example




- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

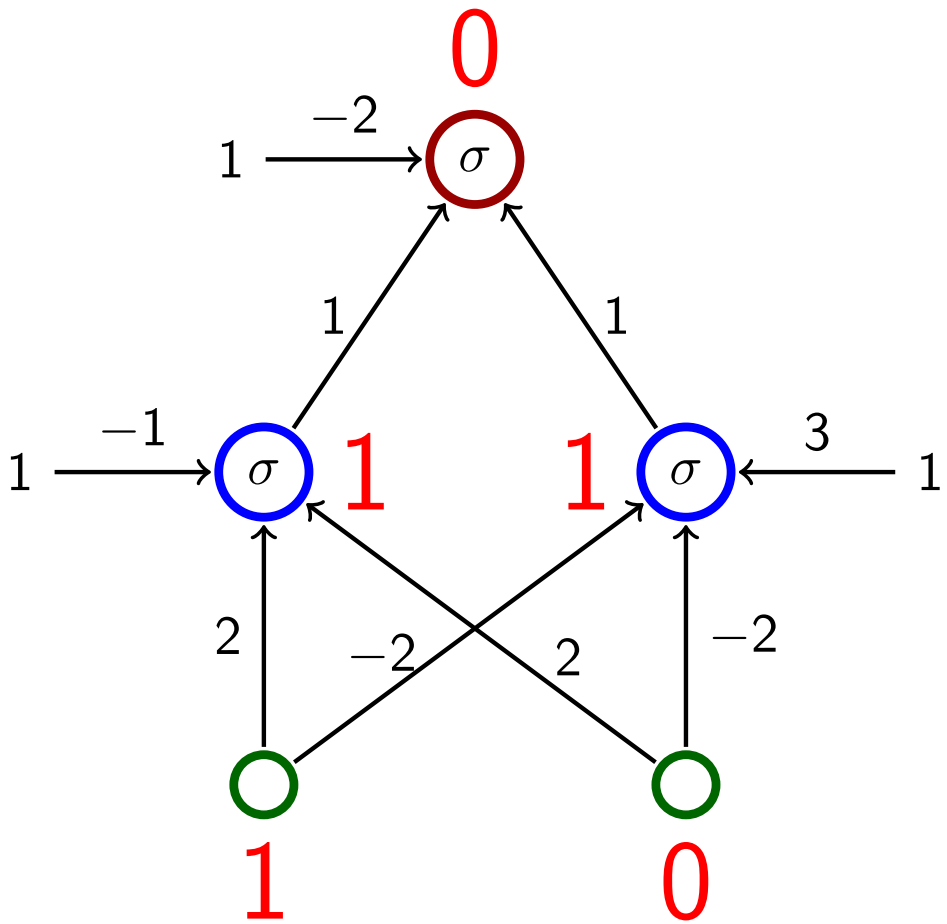
# Example




- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

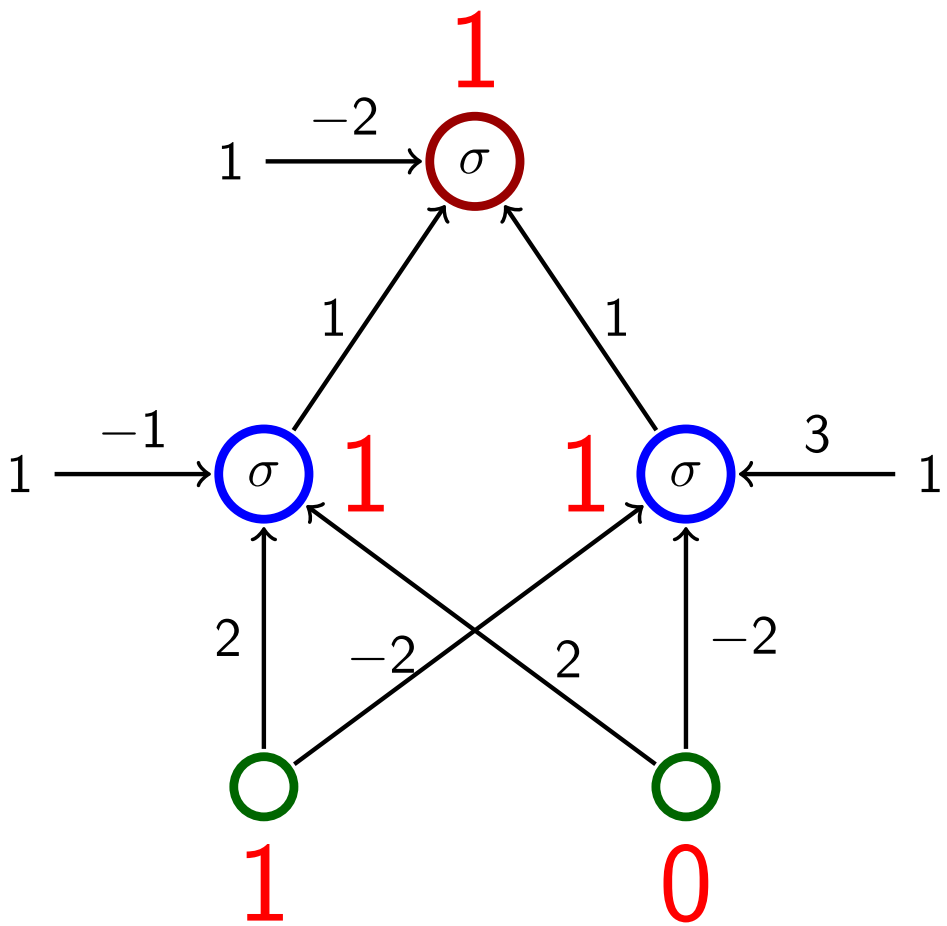
# Example




- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

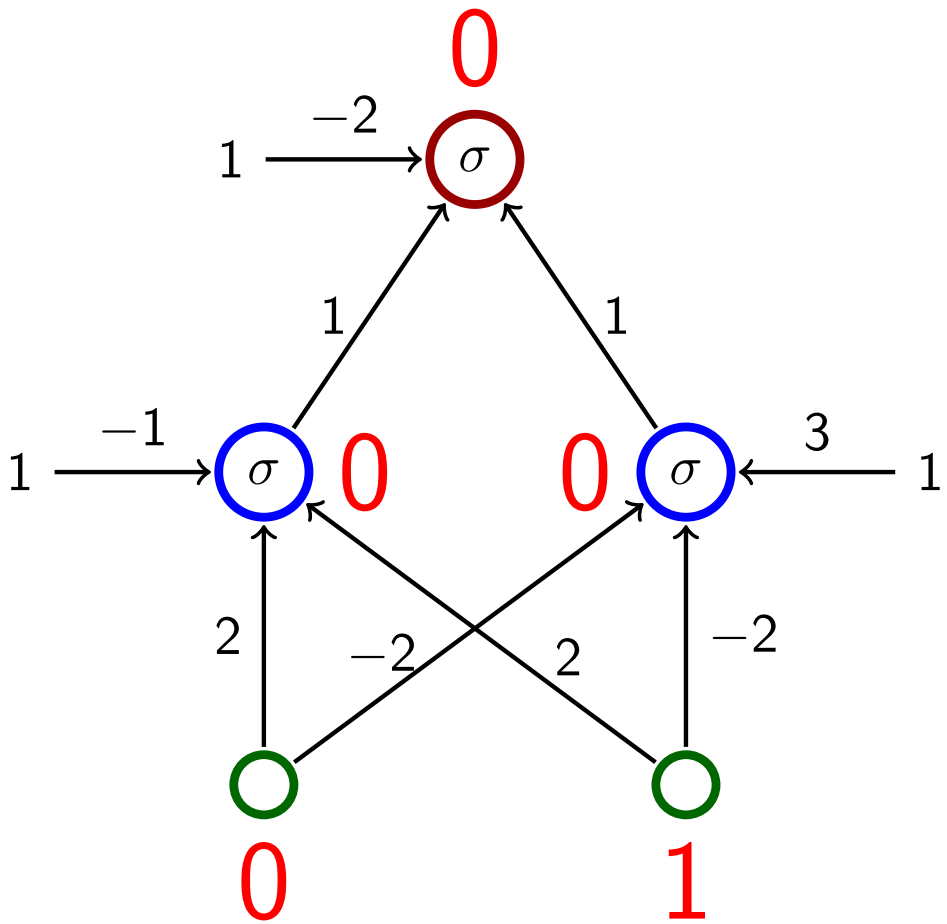
# Example




- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

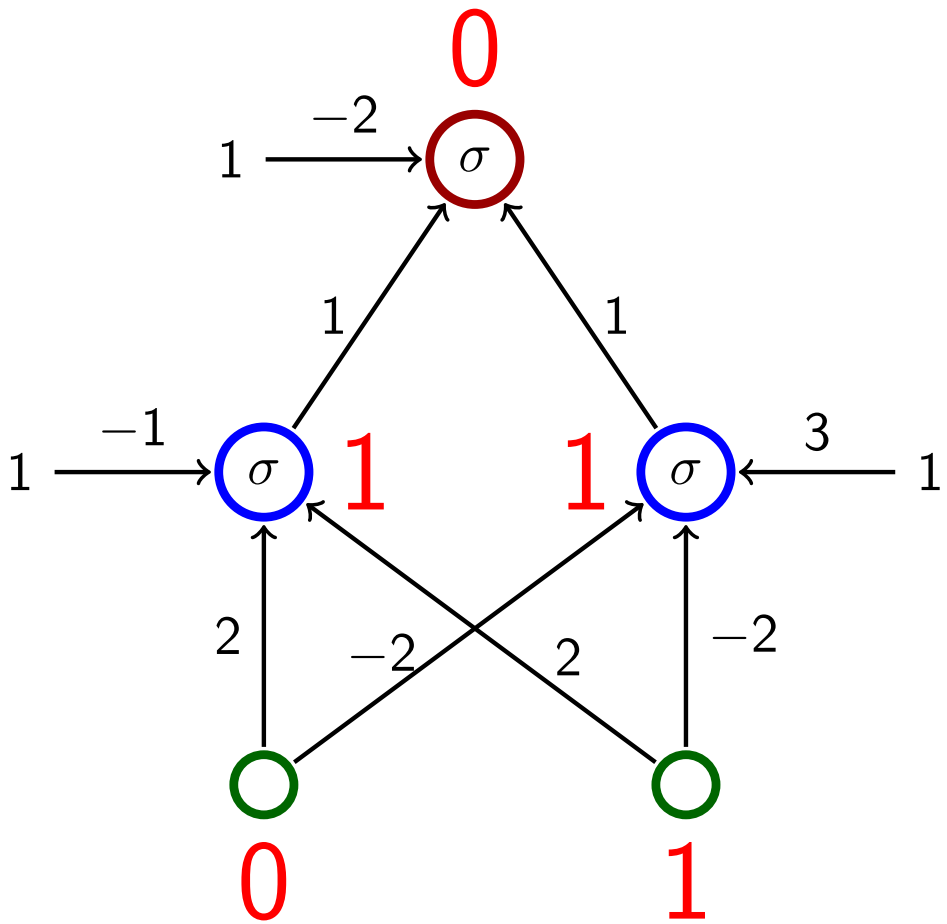
# Example




- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

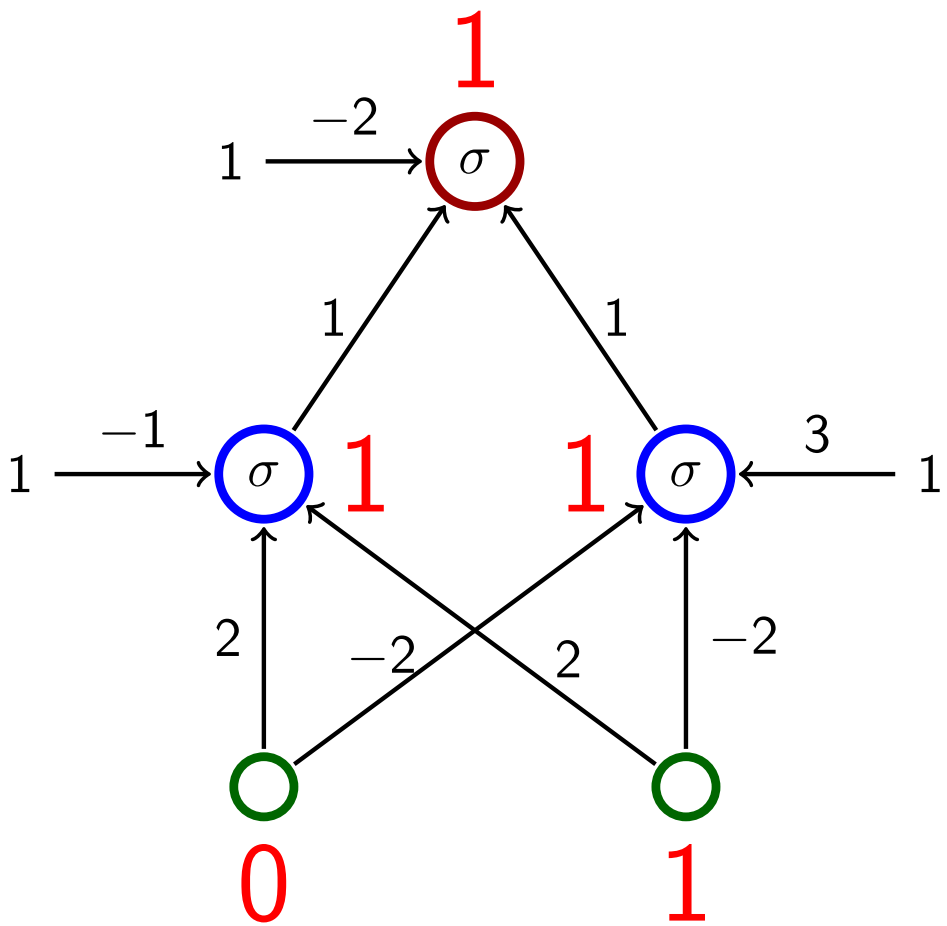
# Example




- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

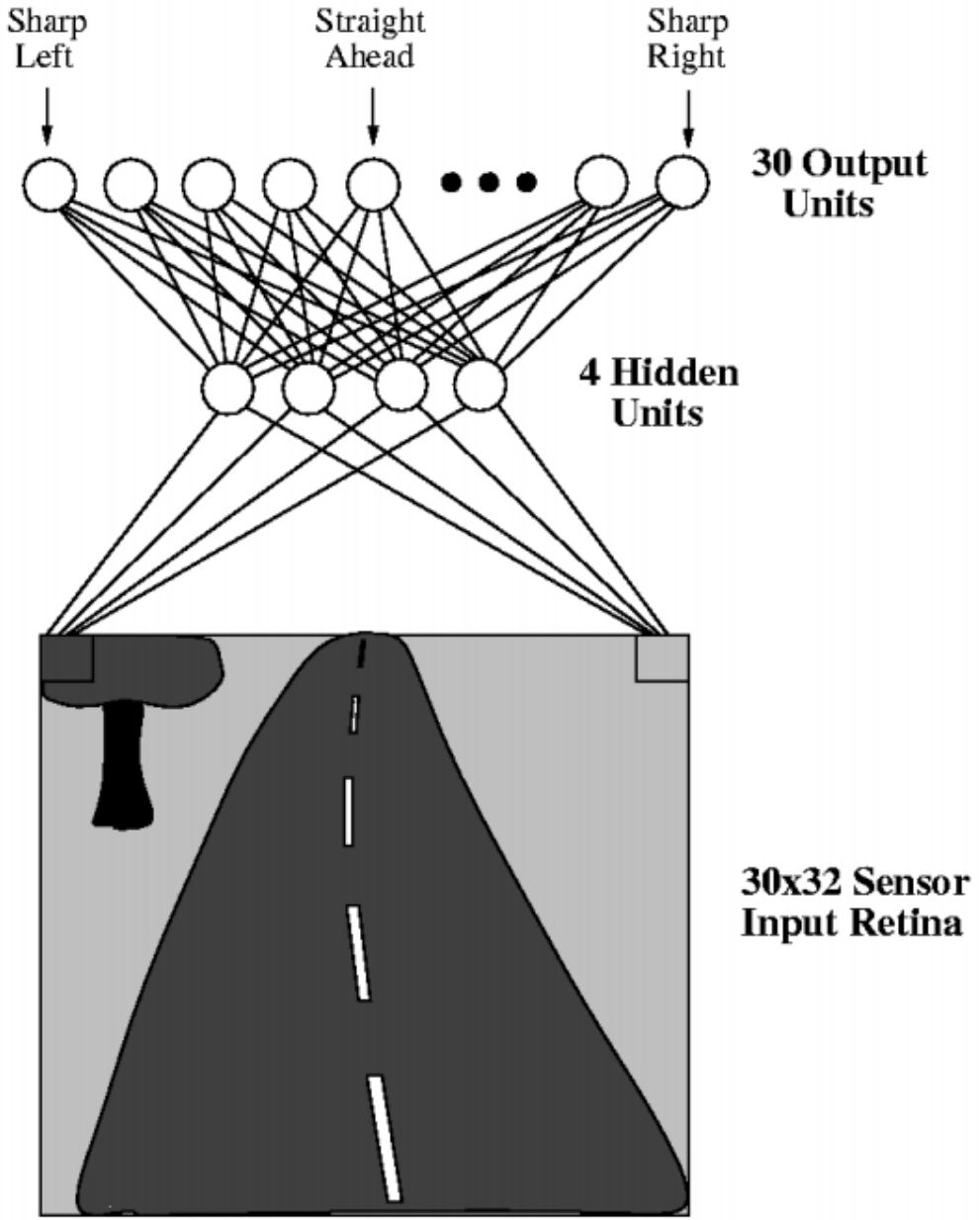
# Example




- Activation function: linear threshold

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

# Classical Example – ALVINN

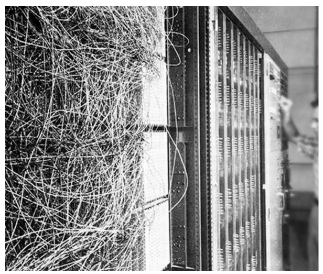



Source: `http://jmvidal.cse.sc.edu/talks/ann/alvin.html`

- One of the first autonomous car driving systems (in the 90s)
- ALVINN drives a car
- The net has $30 \times 32 = 960$ input neurons (the input space is $\mathbb{R}^{960}$).
- The value of each input captures the shade of gray of the corresponding pixel.
- Output neurons indicate where to turn (to the center of gravity).

# A Bit of History

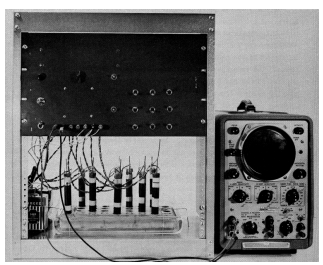
- Perceptron (Rosenblatt et al., 1957)



  - Single layer (i.e., no hidden layers), the activation function *linear threshold*

    (i.e., a bit more general linear classifier)
  - Perceptron learning algorithm
  - Used to recognize digits

- Adaline (Widrow & Hof, 1960)



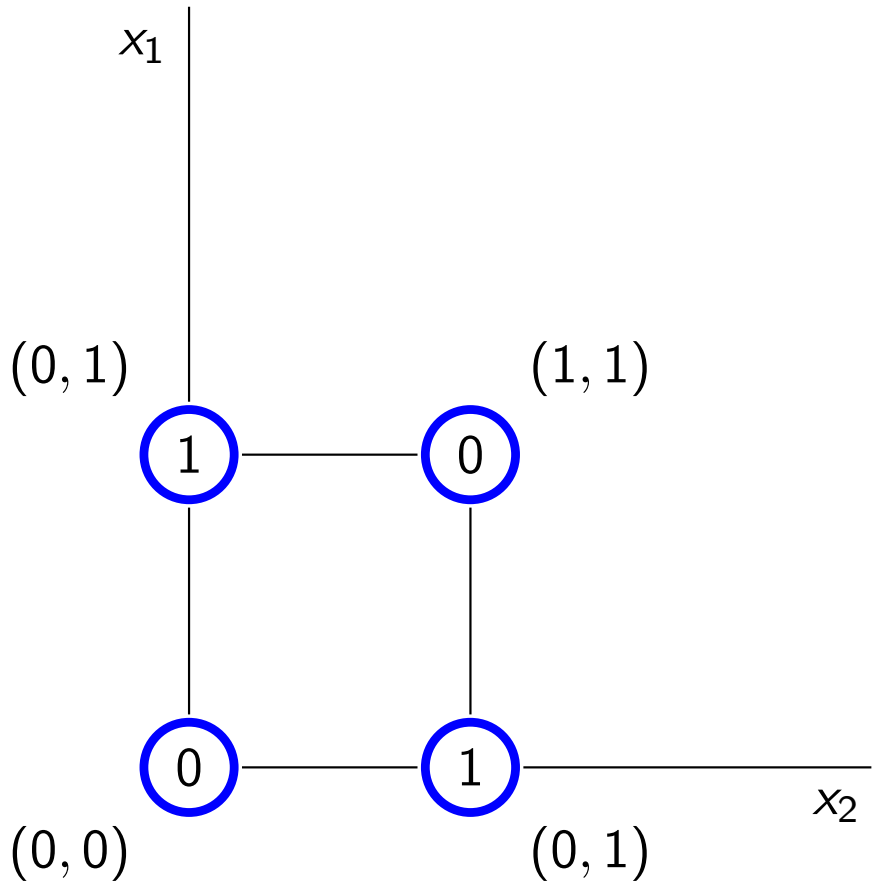
  - Single layer, the activation function *identity*

    (i.e., a bit more linear function)
  - Online version of the gradient descent
  - Used a new circuitry element called *memristor* which was able to "remember" history of current in form of resistance

In both cases, the expressive power is somewhat limited- it can only express linear decision boundaries and linear (affine) functions.

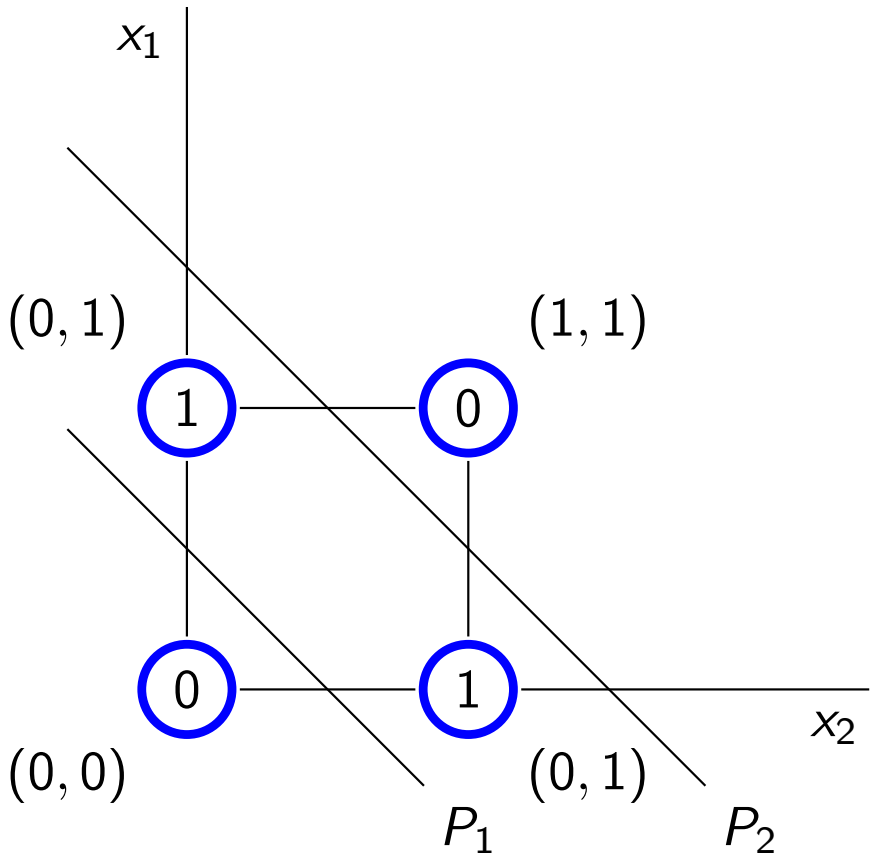
# A Bit of History

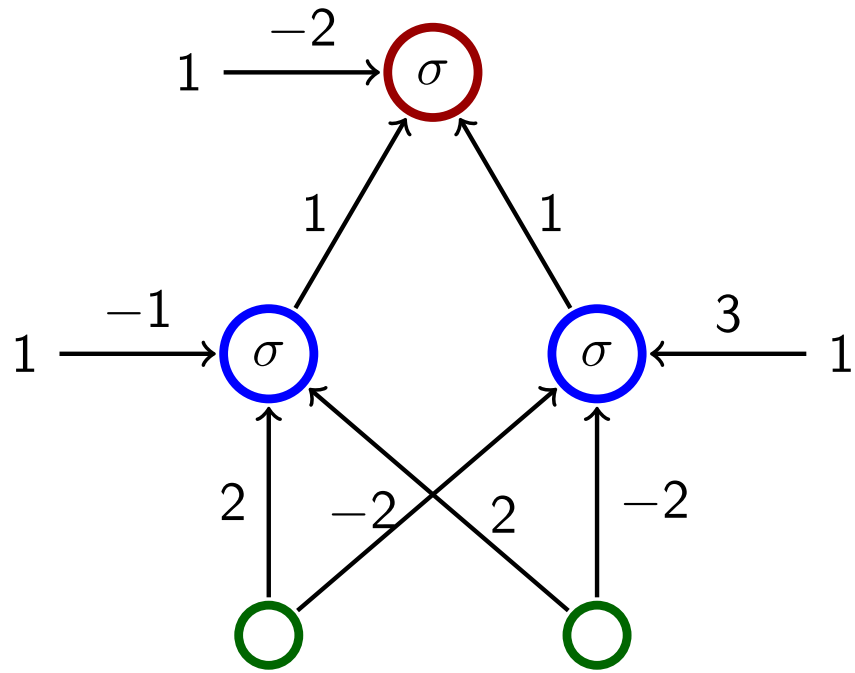
One of the famous (counter)-examples: XOR




No perceptron can distinguish between ones and zeros.

# XOR vs Multilayer Perceptron




(Here, $\sigma$ is a linear threshold function.)

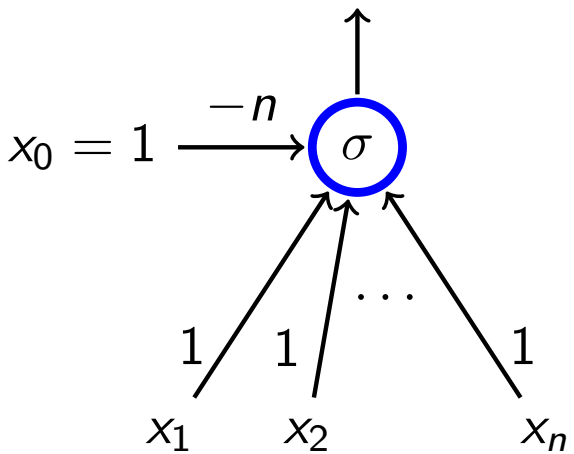
$$P_1 : -1 + 2x_1 + 2x_2 = 0 \qquad P_2 : 3 - 2x_1 - 2x_2 = 0$$

The output neuron performs an intersection of half-spaces.

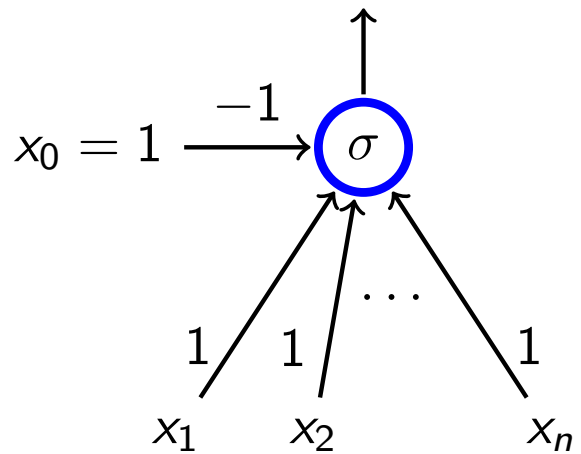
# Boolean functions

Activation function: *unit step function* $\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$

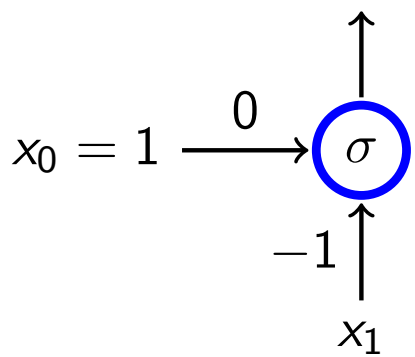
$y = AND(x_1, \ldots, x_n)$



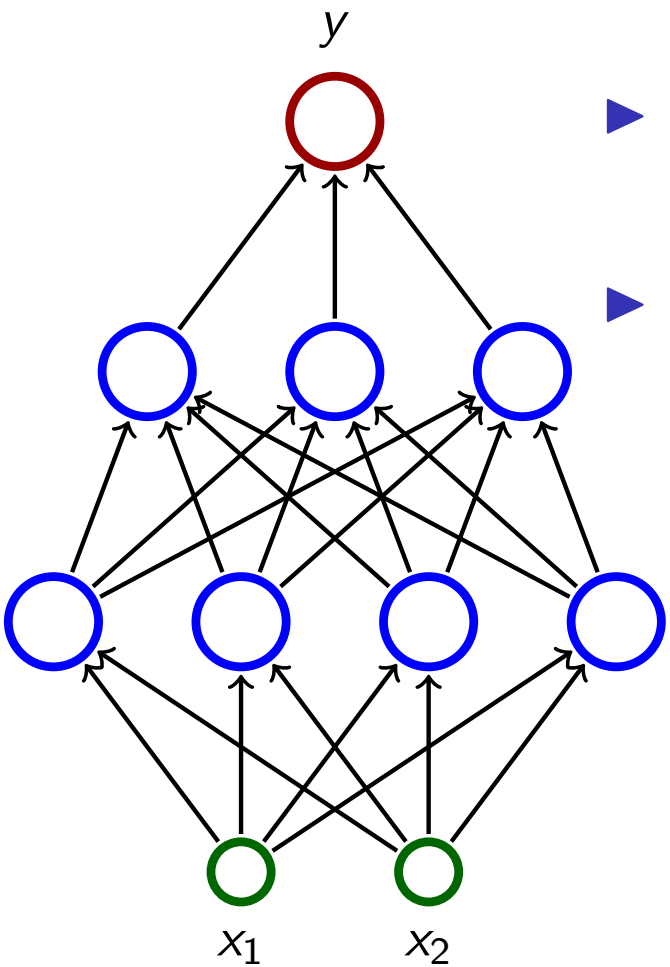

$y = OR(x_1, \ldots, x_n)$




$y = NOT(x_1)$



# Non-linear separation




- Consider a three-layer network; each neuron has the unit step activation function.
- The network divides the input space in two subspaces according to the output (0 or 1).
  - The first (hidden) layer divides the input space into half-spaces.
  - The second layer may, e.g., make intersections of the half-spaces $\Rightarrow$ convex sets.
  - The third layer may, e.g., make unions of some convex sets.

# Example

Consider a triangle $T$ in $\mathbb{R}^2$ determined by three vertices $(-1,-1),(1,-1),(-1,2)$.

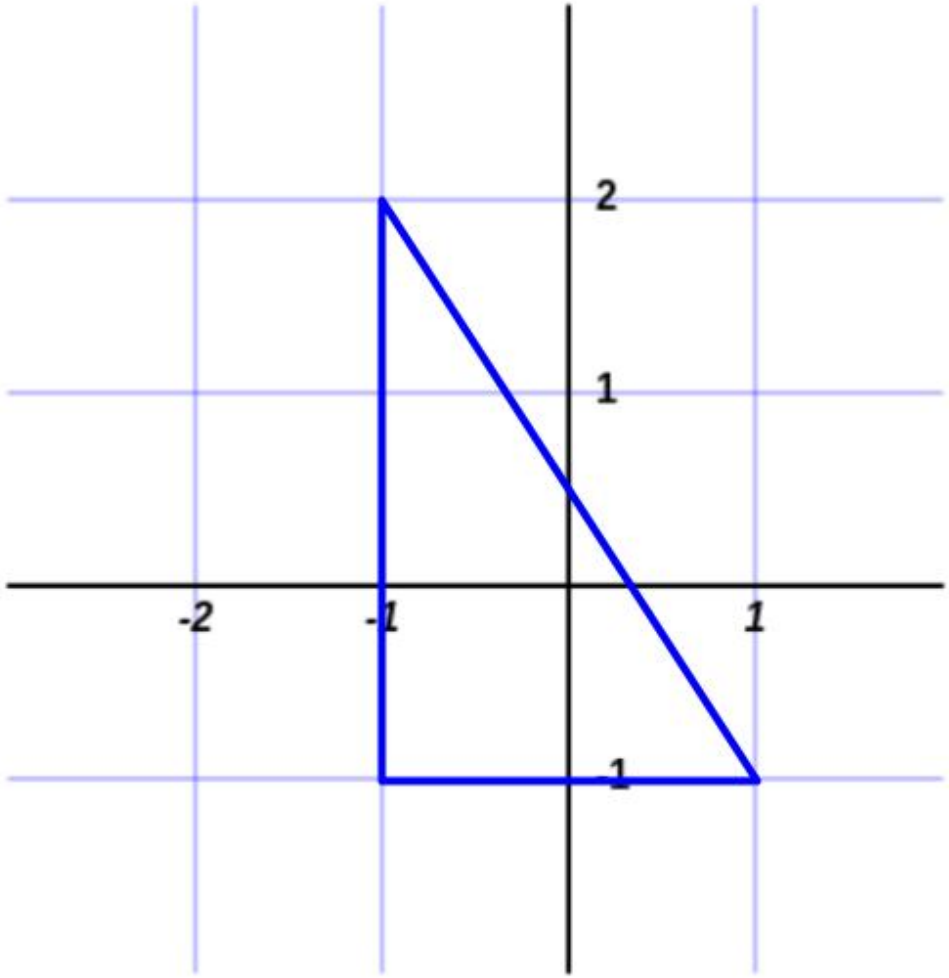
Give an example of a multilayer perceptron (MLP) with two input neurons and a single output neuron computing the function $F:\mathbb{R}^2 \to \{0,1\}$ defined as follows:

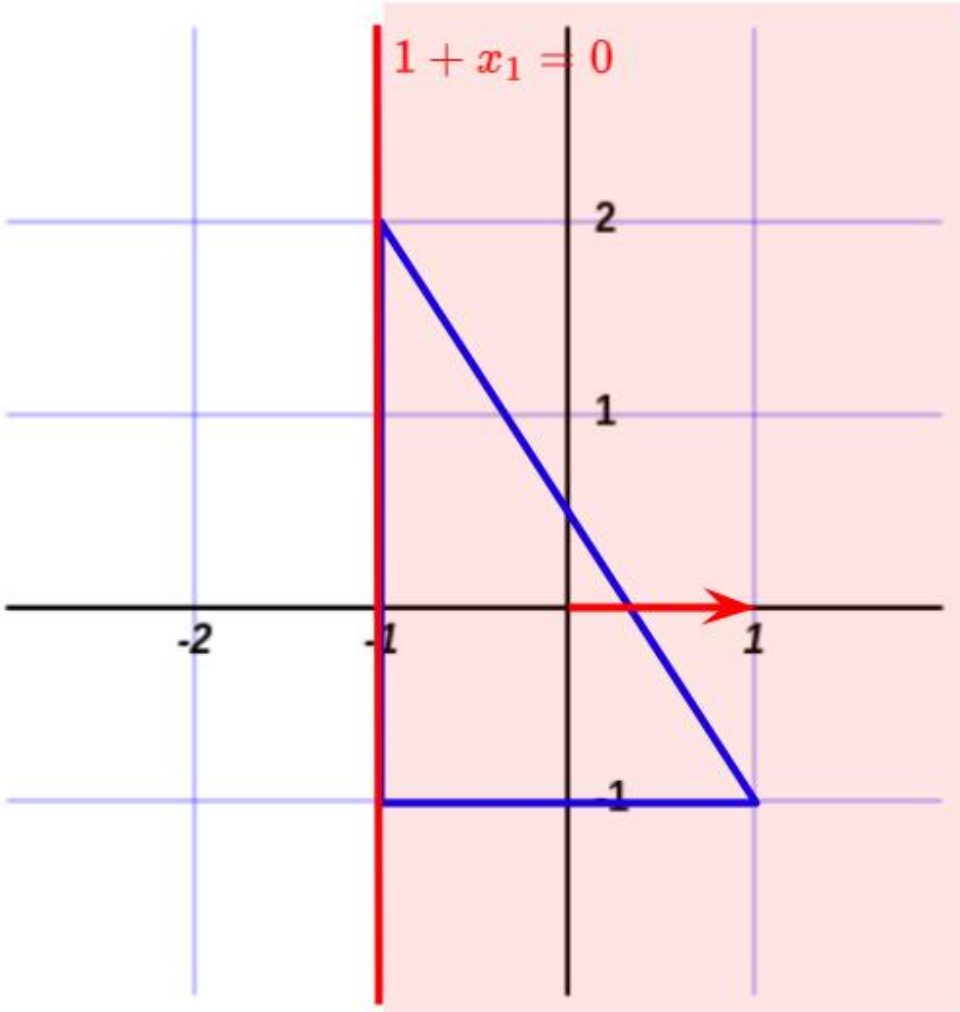
$F(x_1,x_2)=1$ iff $(x_1,x_2)$ lies either inside, or on the border of $T$

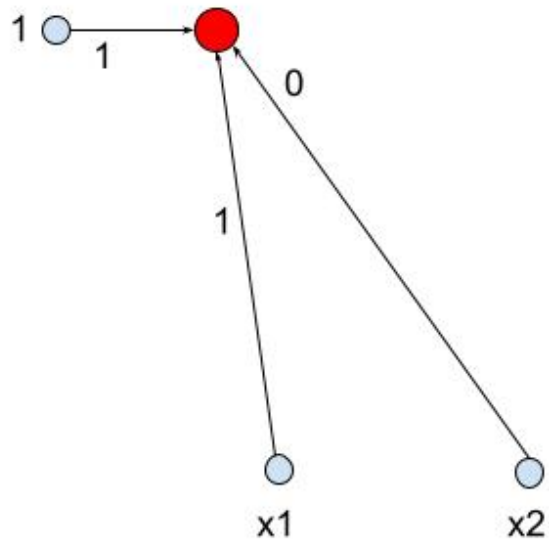
All activation functions in the network should be

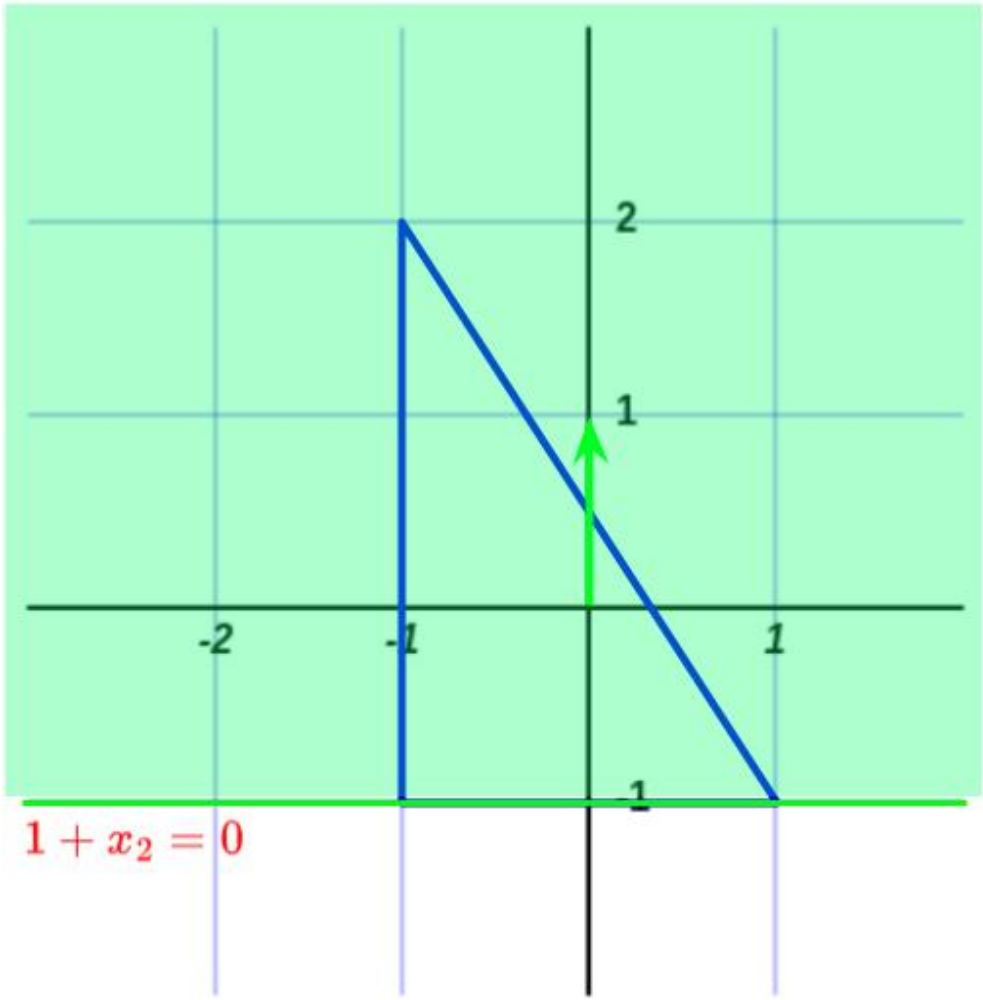
$$\sigma(\xi) = sgn(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

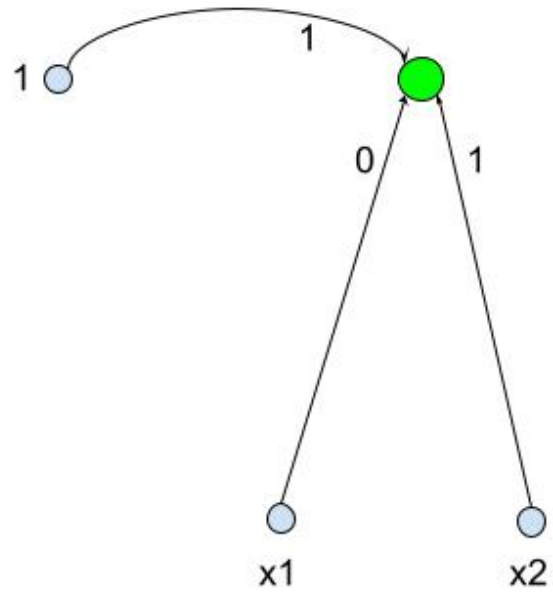
**Homework:** Consider $F(x_1,x_2)=1$ iff $(x_1,x_2)$ lies inside of $T$ (but *not* on the border)

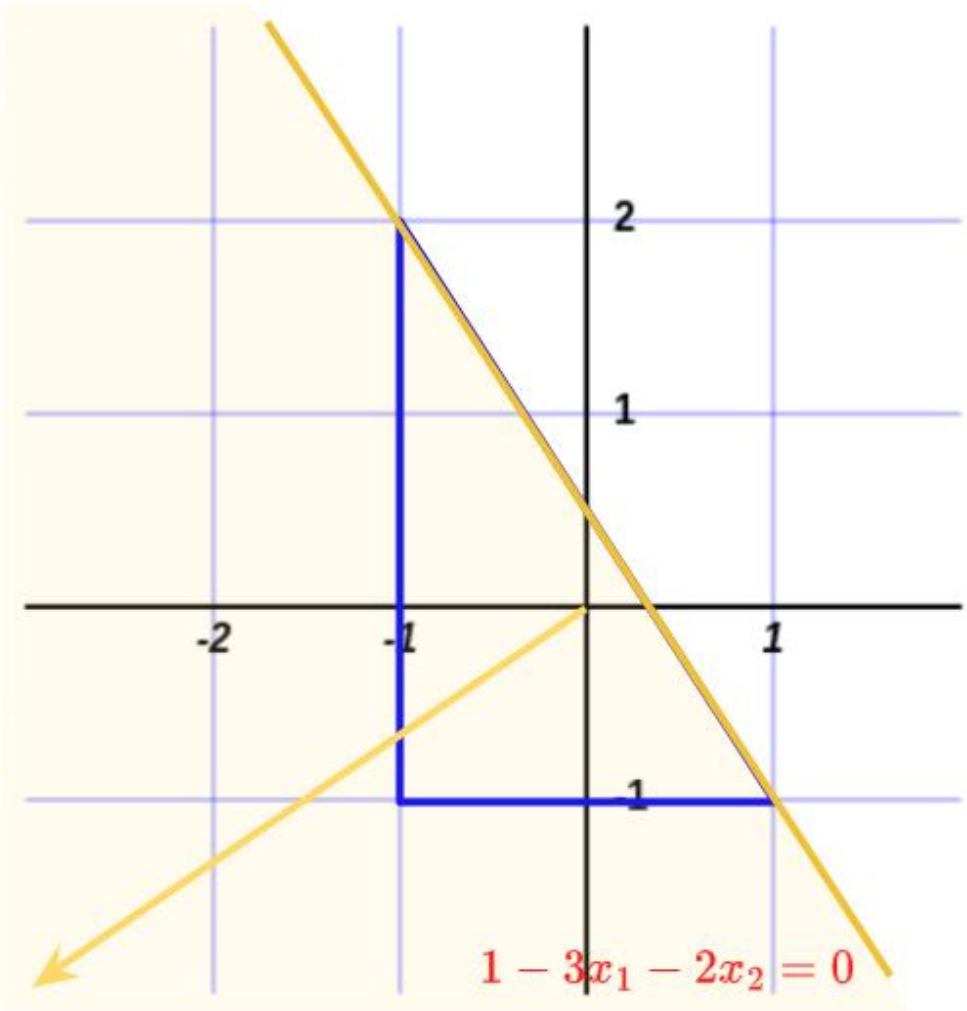

2
1
-2
-1
1
-1

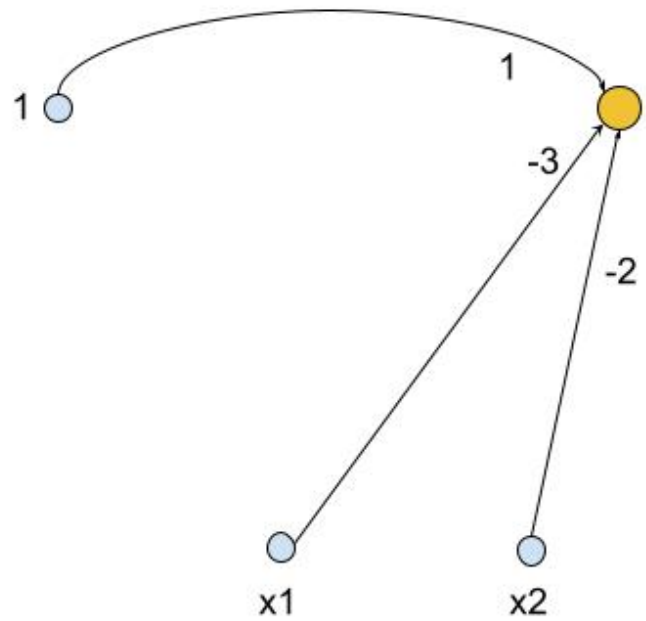

$1 + x_1 = 0$
2
1
-2
-1
1
-1

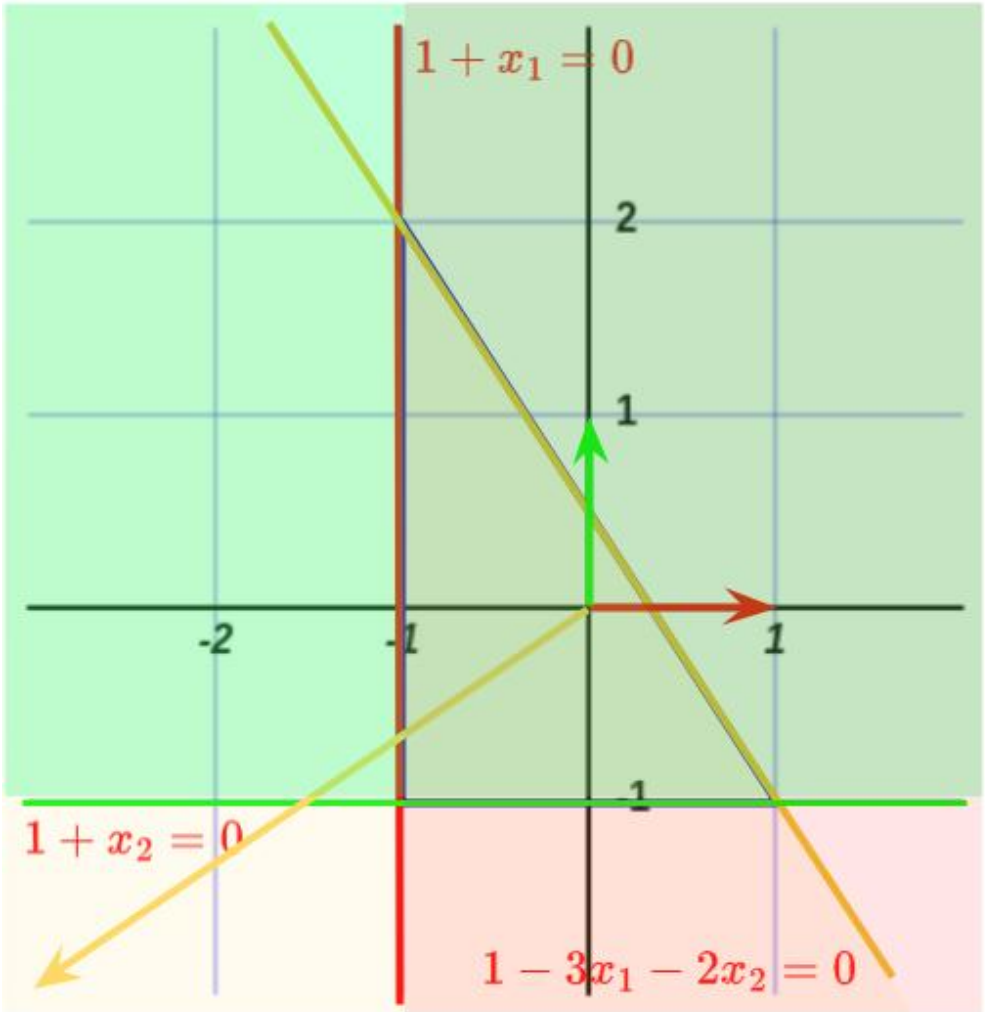

1
1
0
1
x1
x2

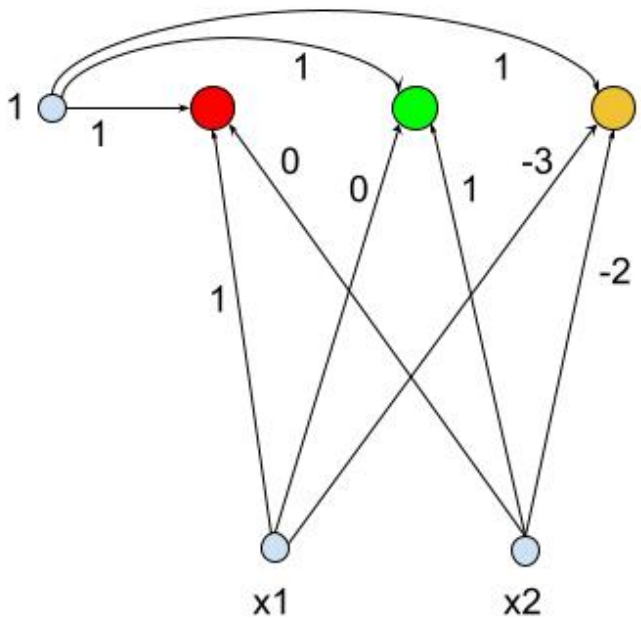

2
1
-2
-1
1
-1
$1 + x_2 = 0$

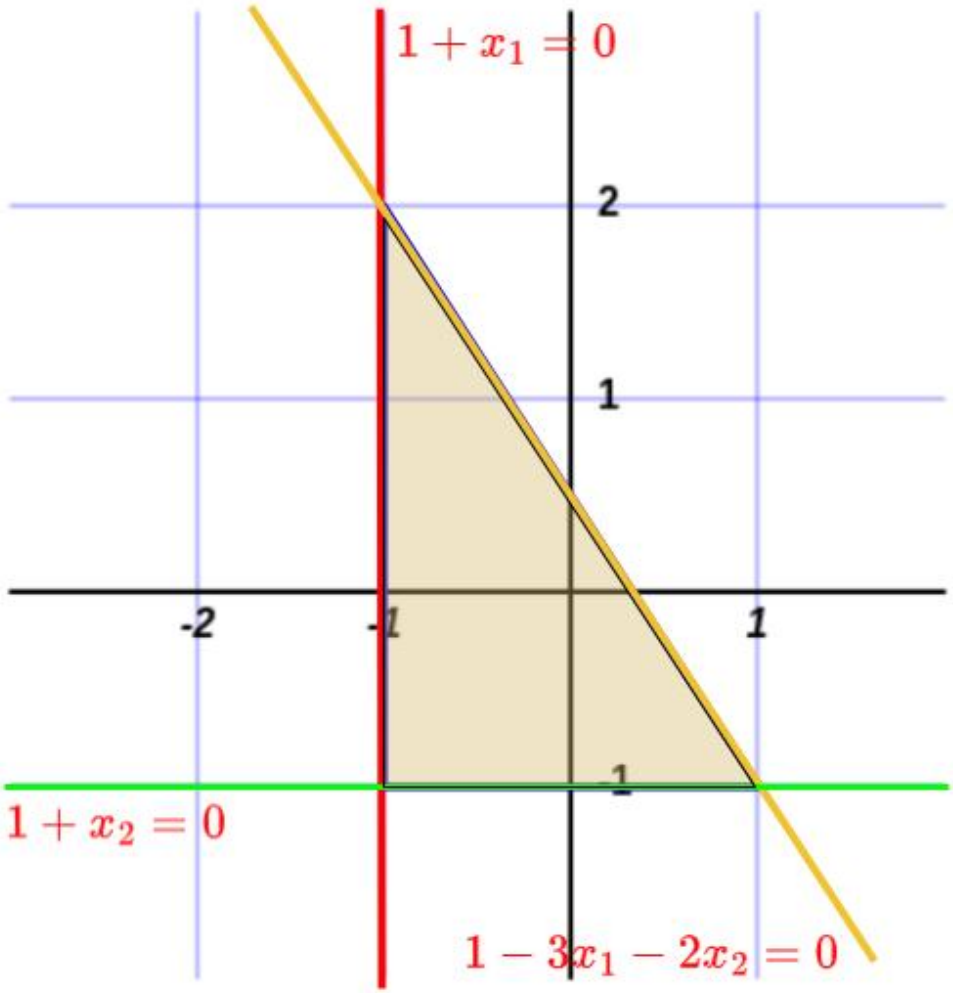

1
1
0
1
x1
x2

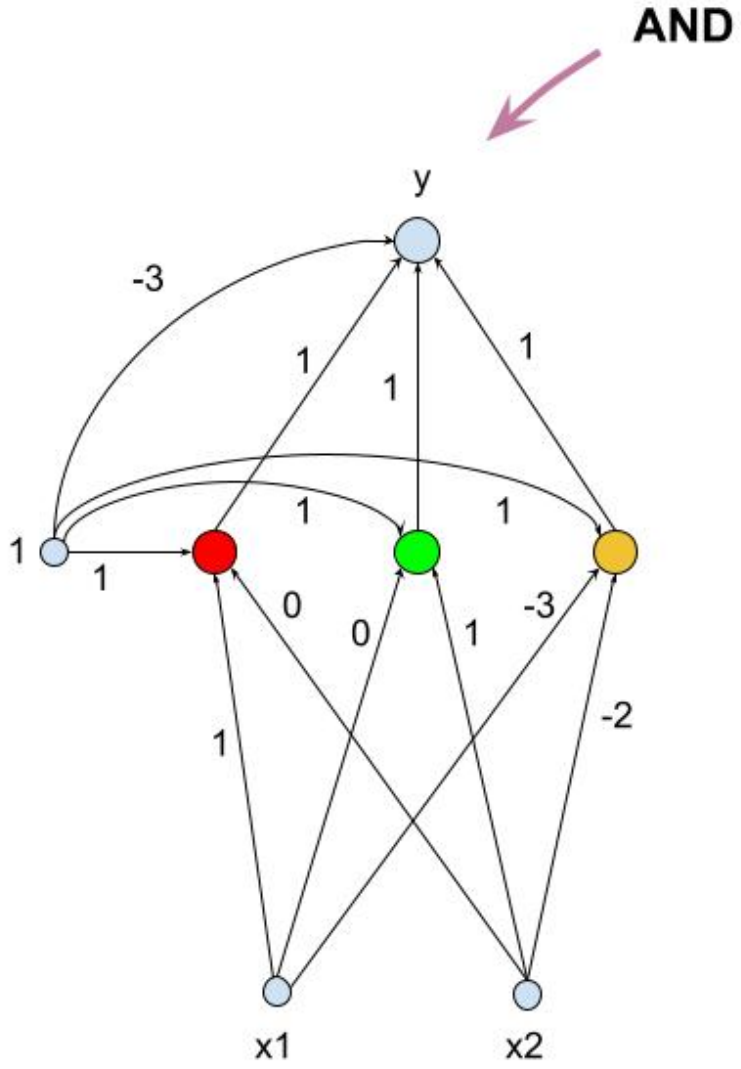

2
1
-2
-1
1
1
$1 - 3x_1 - 2x_2 = 0$


1
1
-3
-2
x1
x2


$1 + x_1 = 0$
$1 + x_2 = 0$
$1 - 3x_1 - 2x_2 = 0$
2
1
-1
-2
-1
1


1
1
1
1
0
0
1
-3
1
-2
x1
x2


$1 + x_1 = 0$
2
1
-2
-1
1
-1
$1 + x_2 = 0$
$1 - 3x_1 - 2x_2 = 0$


AND
y
-3
1
1
1
1
1
1
1
1
0
0
1
-3
-2
1
x1
x2

# Expressive Power of MLP

Cybenko's theorem:

- Two-layer networks with a single output neuron and a single layer of hidden neurons (with the logistic sigmoid as the activation function) can
  - approximate with arbitrarily small error any "reasonable" boundary
    a given input is classified as 1 iff the output value of the network is $\geq 1/2$.
  - approximate with arbitrarily small error any "reasonable" function from $[0, 1]$ to $(0, 1)$.

  Here, "reasonable" means that it is pretty tough to find a function that is not reasonable.

So, multilayer perceptrons are sufficiently powerful for any application.

But for a long time, at least throughout the 60s and 70s, nobody well-known knew any efficient method for training multilayer networks!

An efficient way of using the gradient descent was published in 1986!

# MLP – Notation

- $X$ set of input neurons
- $Y$ set of output neurons
- $Z$ set of all neurons (tedy $X, Y \subseteq Z$)

- individual neurons are denoted by indices, e.g., $i, j$.
- $\xi_j$ is the inner potential of the neuron $j$ when the computation is finished.
- $y_j$ is the output value of the neuron $j$ when the computation is finished.

  (we formally assume $y_0 = 1$)
- $w_{ji}$ is the weight of the arc **from** the neuron $i$ **to** the neuron $j$.

- $j_{\leftarrow}$ is the set of all neurons from which there are edges to $j$

  (i.e. $j_{\leftarrow}$ is the layer directly below $j$)
- $j^{\rightarrow}$ is the set of all neurons with edges from $j$.

  (i.e. $j^{\rightarrow}$ is the layer directly above $j$)

# MLP – Notation

- Inner potential of a neuron $j$:

$$\xi_j = \sum_{i \in j_{\leftarrow}} w_{ji} y_i$$

- A value of a non-input neuron $j \in Z \setminus X$ when the computation is finished is

$$y_j = \sigma_j(\xi_j)$$

  Here $\sigma_j$ is an activation function of the neuron $j$.

  ($y_j$ is determined by weights $\vec{w}$ and a given input $\vec{x}$, so it's sometimes written as $y_j[\vec{w}](\vec{x})$ )

- Fixing weights of all neurons, the network computes a function $F[\vec{w}] : \mathbb{R}^{|X|} \to \mathbb{R}^{|Y|}$ as follows: Assign values of a given vector $\vec{x} \in \mathbb{R}^{|X|}$ to the input neurons, evaluate the network, then $F[\vec{w}](\vec{x})$ is the vector of values of the output neurons.

  Here, we implicitly assume a fixed ordering on input and output vectors.

# MLP – Learning

- Given a set $D$ of training examples:

$$D = \left\{ \left( \vec{x}_k, \vec{d}_k \right) \quad | \quad k = 1, \ldots, p \right\}$$

  Here $\vec{x}_k \in \mathbb{R}^{|X|}$ and $\vec{d}_k \in \mathbb{R}^{|Y|}$. We write $d_{kj}$ to denote the value in $\vec{d}_k$ corresponding to the output neuron $j$.

- **Error Function:** $E(\vec{w})$ where $\vec{w}$ is a vector of all weights in the network. The choice of $E$ depends on the solved task (classification vs regression etc.).
  **Example (Squared error):**

$$E(\vec{w}) = \sum_{k=1}^{p} E_k(\vec{w})$$

  where

$$E_k(\vec{w}) = \frac{1}{2} \sum_{j \in Y} (y_j[\vec{w}](\vec{x}_k) - d_{kj})^2$$

# MLP – Batch Gradient Descent

The algorithm computes a sequence of weights $\vec{w}^{(0)}, \vec{w}^{(1)}, \ldots$.

- weights $\vec{w}^{(0)}$ are initialized randomly close to 0
- in the step $t+1$ (here $t = 0, 1, 2 \ldots$) is $\vec{w}^{(t+1)}$ computed as follows:

$$w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)}$$

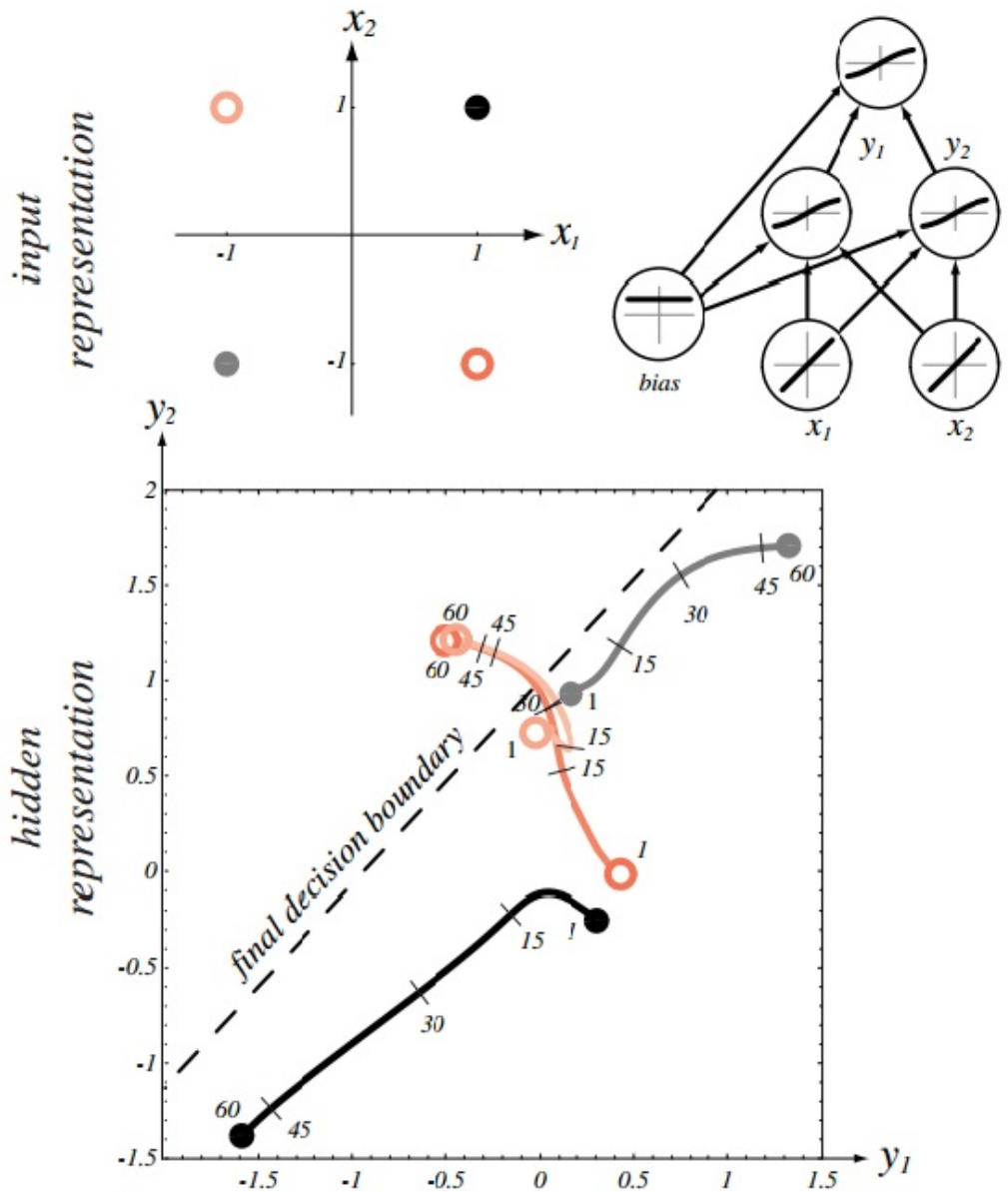
where

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t)})$$

is the weight change $w_{ji}$ and $0 < \varepsilon(t) \leq 1$ is the learning rate in the step $t+1$.

Note that $\frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t)})$ is a component of $\nabla E$, i.e., the weight change in the step $t+1$ can be written as follows: $\vec{w}^{(t+1)} = \vec{w}^{(t)} - \varepsilon(t) \cdot \nabla E(\vec{w}^{(t)})$.

# Illustration of Gradient Descent – XOR




Source: Pattern Classification (2nd Edition); Richard O. Duda, Peter E. Hart, David G. Stork

# Stochastic Gradient Descent (SGD)

Assume that $E(\vec{w}) = \sum_{k=1}^{p} E_k(\vec{w})$ where $E_k(\vec{w})$ is an error w.r.t. the single training example $(\vec{x}_k, \vec{d}_k)$.

- weights in $\vec{w}^{(0)}$ are randomly initialized to values close to 0
- in the step $t+1$ (here $t = 0, 1, 2 \ldots$), weights $\vec{w}^{(t+1)}$ are computed as follows:
  - Choose (randomly) a set of training examples $T \subseteq \{1, \ldots, p\}$
  - Compute

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} + \Delta\vec{w}^{(t)}$$

where

$$\Delta\vec{w}^{(t)} = -\varepsilon(t) \cdot \sum_{k \in T} \nabla E_k(\vec{w}^{(t)})$$

- $0 < \varepsilon(t) \leq 1$ is a *learning rate* in step $t+1$
- $\nabla E_k(\vec{w}^{(t)})$ is the gradient of the error of the example $k$

Note that the random choice of the minibatch is typically implemented by randomly shuffling all data and then choosing minibatches sequentially.

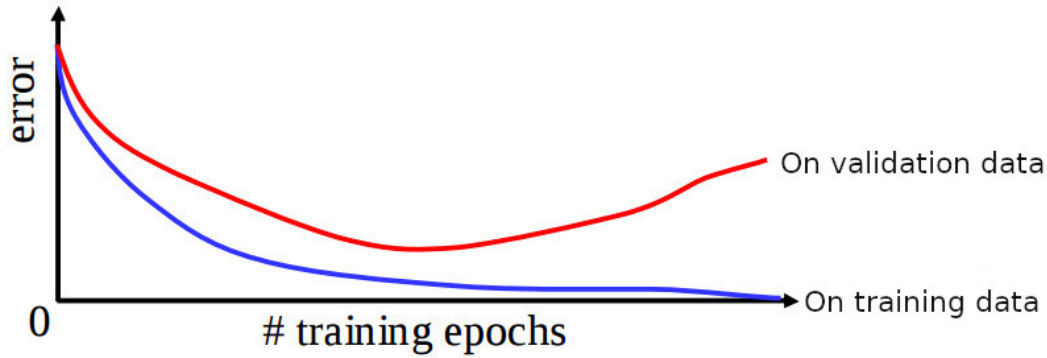
# Comments on Training Algorithm

- Not guaranteed to converge to zero training error, may converge to a local minimum or oscillate indefinitely.
- In practice, does converge to low error for many large networks on (big) real data.
- Many epochs (thousands) may be required, hours or days of training for large networks.

There are many issues concerning learning efficiency (data normalization, selection of activation functions, weight initialization, learning rate, efficiency of the gradient descent itself, etc.) – see PV021.

# Overfitting

- Due to their expressive power, neural networks are quite sensitive to overfitting.




- Keep a hold-out validation set and test the error of the network on this set after every epoch. Stop training when additional epochs increase the validation error.

  The validation error can be measured by completely different means than the training error $E$.

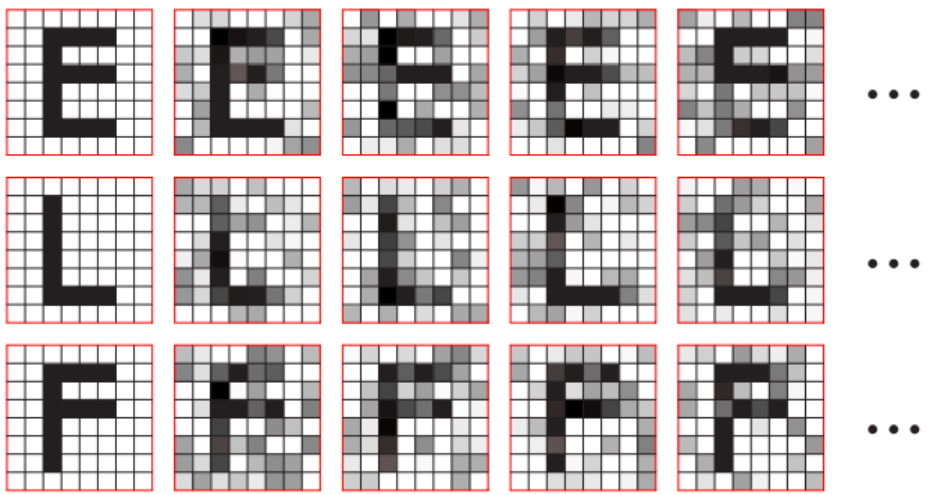
# Hidden Neurons Representations

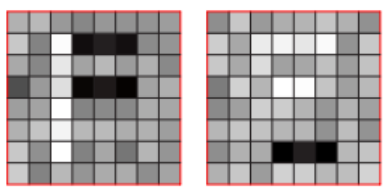
Trained hidden neurons can be seen as newly constructed features.

E.g., in a two-layer network used for classification, the hidden layer transforms the input so that important features become explicit (and hence the result may become linearly separable).

Consider a two-layer MLP, 64-2-3, for classifying letters (three output neurons, each corresponding to one of the letters).

*sample training patterns*





*learned input-to-hidden weights*

# Optimal Architecture?

- For MLP: Too few hidden neurons prevent the network from adequately fitting the data. Too many hidden units can result in overfitting.

  (There are advanced methods that prevent overfitting even for rich models, such as regularization, where the error function penalizes overfitting – see PV021.)
- There are (almost) infinitely many types of architectures of neural networks (convolutional, recurrent, transformers, adversarial, etc.) suitable for various tasks.
- **Transfer learning:** Start with a known solution to a related problem.

  Simplified view: Preserve lower parts of the network trained to solve the related problem (feature extractors). Add your top part and then train only the new top part (or train the whole network but carefully).

# How to Choose Activation Functions & Error

- **Hidden neurons:** "Almost" linear activations such as (leaky) ReLU ($y = \max(0, \xi)$)

  Better than sigmoidal that saturate more often.
- **Output neurons:** Single output:
  - Regression: Typically "linear" output, i.e., no activation on the output neuron.
  - Binary classification: Logistic sigmoid $y = 1/(1 + e^{-\xi})$
- **Error:** Single output:
  - Regression: (Mean) squared error
  - Binary classification: Binary cross-entropy

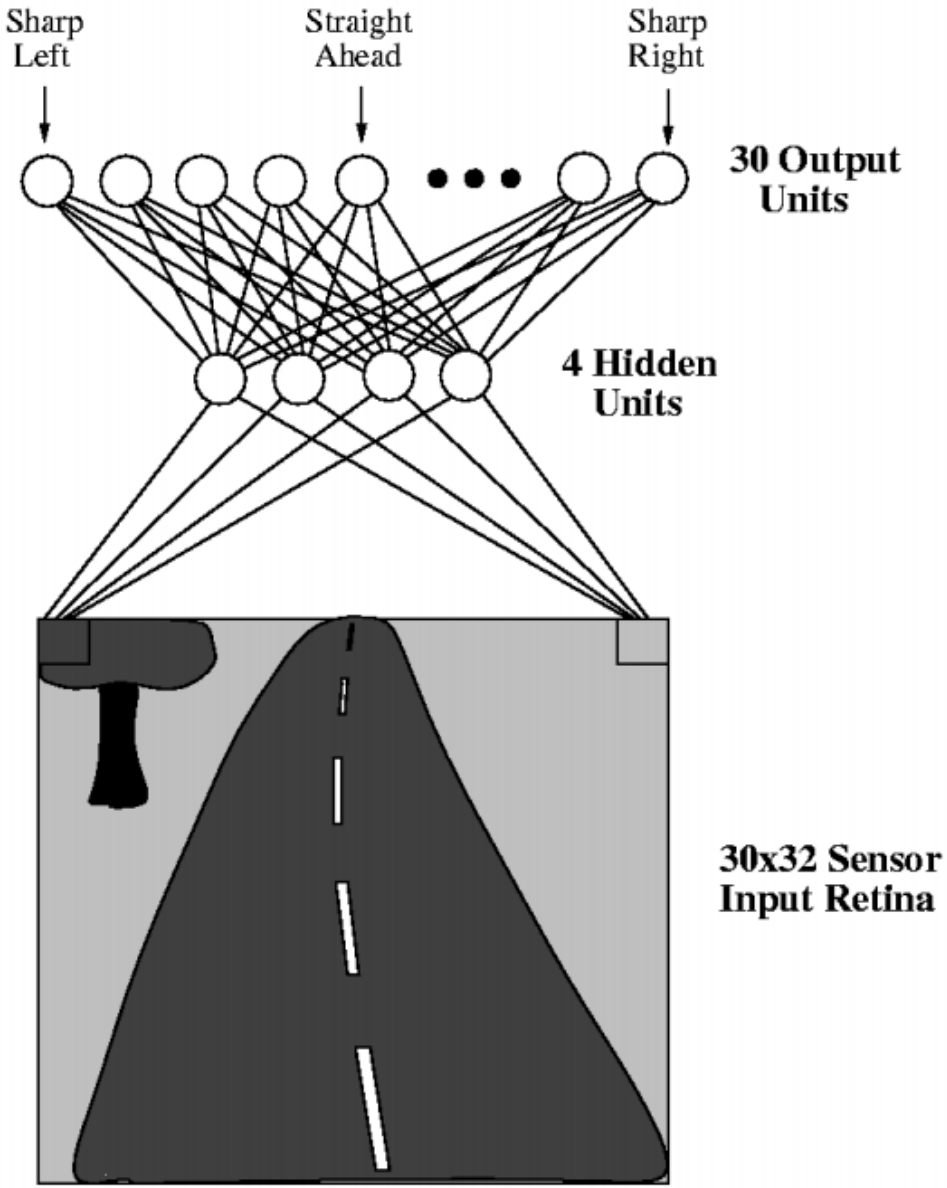
For multiple outputs and classification, use softmax output and cross-entropy.

# Applications

- Image recognition, segmentation, etc.
- Machine translation and other text processing
- Text generation, image generation, movie generation, theatre plays generation
- Text to Speech and vice versa
- Finance, business predictions, fraud detection
- Game playing (backgammon is a classic example, AlphaGo is the famous one, computer games are the big ones, **bridge** is the hard one)
- (artificial brain and intelligence)
- ...

Text and image processing are possibly the most advanced deep learning applications.

# ALVINN


Sharp Left
Straight Ahead
Sharp Right
30 Output Units
4 Hidden Units
30x32 Sensor Input Retina

# ALVINN

- Two layer MLP, $960 - 4 - 30$ (sometimes $960 - 5 - 30$)
- Inputs correspond to pixels.
- Sigmoidal activation function (logistic sigmoid).
- Direction corresponds to the center of gravity.

  , I.e., output neurons are considered as points of mass evenly distributed along a line. The weight of each neuron corresponds to its value.

# ALVINN – Training

Trained while driving.

- A camera captured the road from the front window, approx. 25 pictures per second
- Training examples $(\vec{x}_k, \vec{d}_k)$ where
  - $\vec{x}_k$ = image of the road
  - $\vec{d}_k \approx$ corresponding direction of the steering wheel set by the driver
- the values $\vec{d}_k$ computed using Gaussian distribution:

$$d_{ki} = e^{-D_i^2/10}$$

Where $D_i$ is the distance between the $i$-th output from the one corresponding to the steering wheel's real direction.

(The authors claimed this approach is better than the binary output because similar road directions induce similar driver reactions.)

# Selection of Training Examples

Naive approach: just take images from the camera.

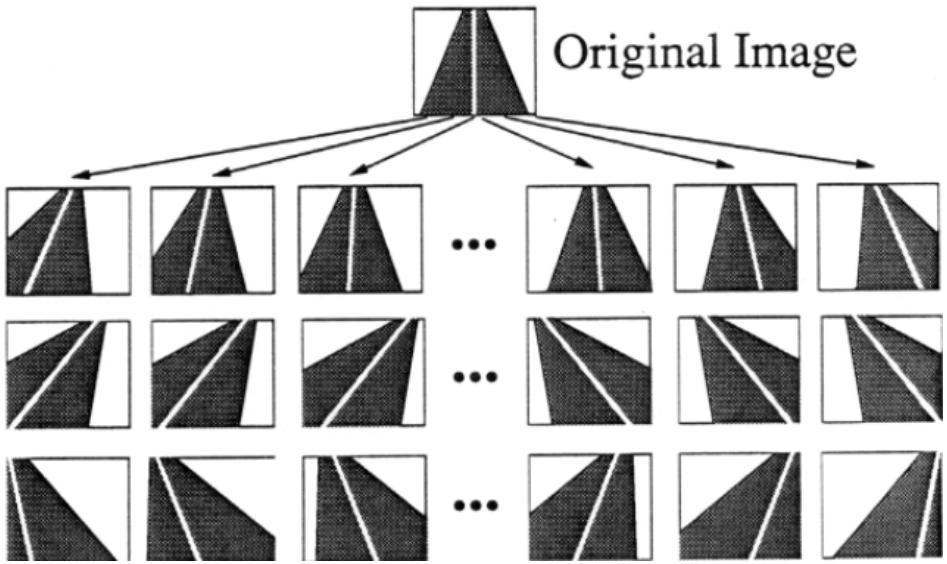
Problems:

- A too-good driver never teaches the network how to solve deviations from the right track. A couple of harsh solutions:
  - turn the learning off momentarily, deviate from the right track, then turn on the learning and let the network learn how to solve the situation.
  - let the driver go crazy! (a bit dangerous, expensive, unreliable)
- Images are very similar (the network sees the road from the right lane), overfitting.

# Selection of Training Examples

Problems with too good driver were solved as follows:

- every image of the road has been has been transformed to 15 slightly different copies




The repetitiveness of images was solved as follows:

- the system has a buffer of 200 images (including the 15 copies of the current one), in every round trains on these images
- afterward, a new image is captured, 15 copies made, and these new 15 substitute 15 selected from the buffer (10 with the smallest training error, five randomly)

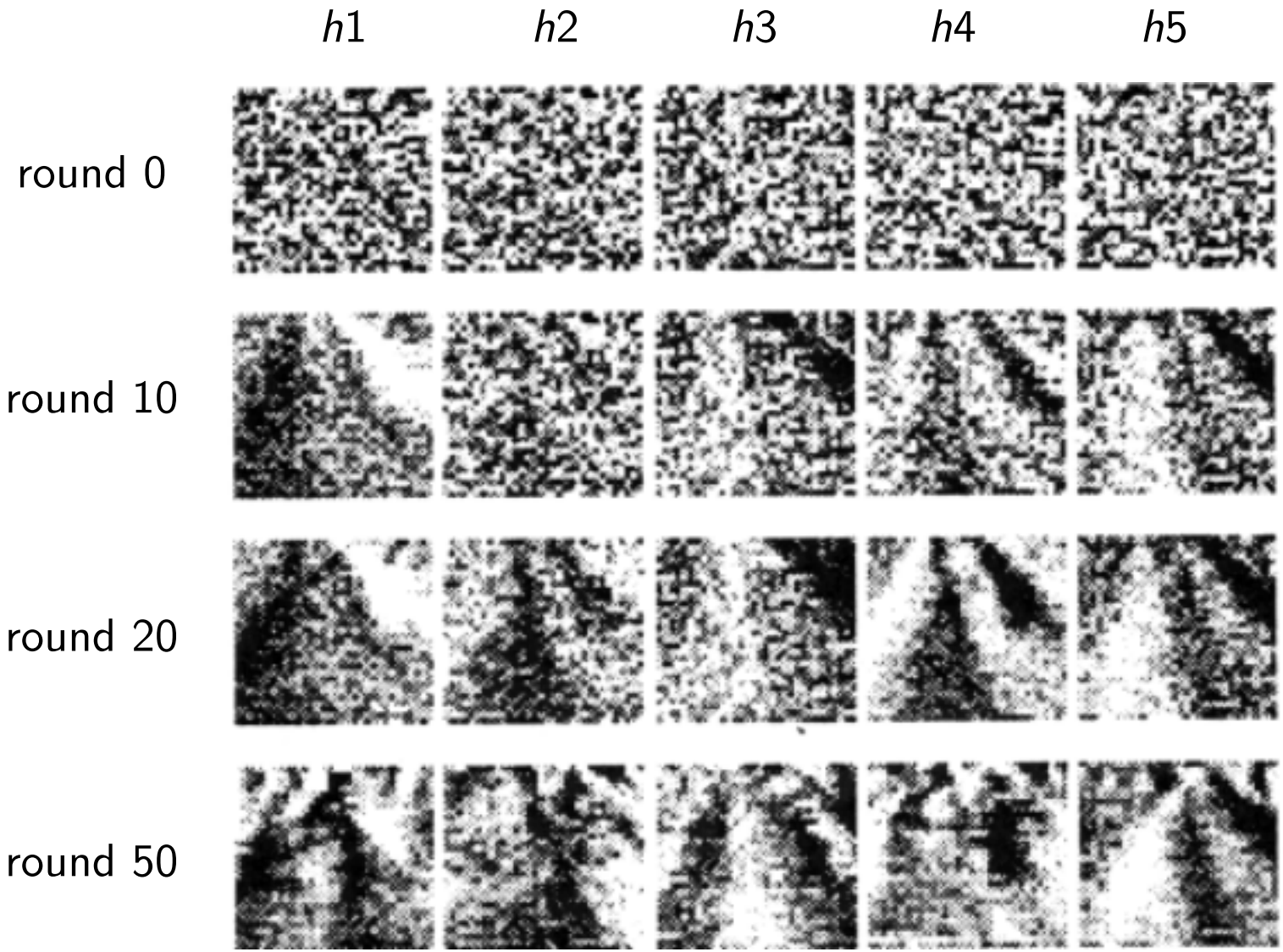
# ALVINN – Training

- gradient descent
- constant learning rate (possibly different for each neuron – see PV021)
- some other optimizations (see PV021)

The result:

- Training took 5 minutes, and the speed was 4 miles per hour
  (The speed was limited by the hydraulic controller of the steering wheel, not the learning algorithm.)
- ALVINN was able to go through roads it had never "seen" and in different weather

# ALVINN – Weight Learning




Here $h1, \ldots, h5$ are values of hidden neurons.

# Backpropagation

# How to Compute the Gradient?

To implement a single step of the gradient descent, we need to compute the partial derivatives $\frac{\partial E}{\partial w_{ij}}$ of $E$ w.r.t. all weights $w_{ij}$.

**To simplify** consider for now a restricted case:

- Single element training set

$$D = \{(x, d)\}$$

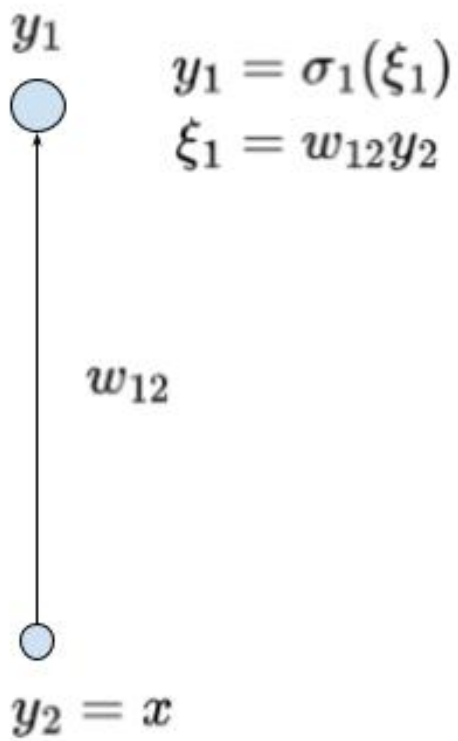
where $x \in \mathbb{R}$ and $d \in \mathbb{R}$ are numbers
(i.e., not vectors as in the general case).

- The error function is the squared error.
  - Assume that 1 is the output neuron,
  - which means that $y_1 = y_1[\vec{w}](x)$ is the output of the network with weights $\vec{w}$ and the input $x$,
  - and the error on $D$ is then

$$E(\vec{w}) = \frac{1}{2}(y_1 - d)^2$$

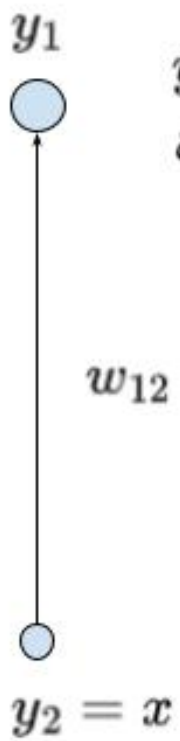
$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$

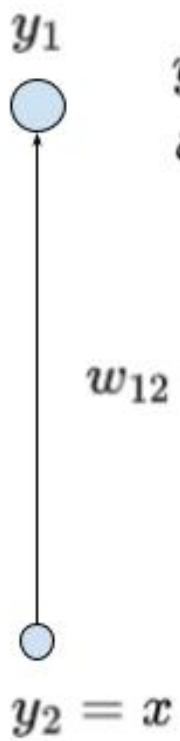



$$y_1 = \sigma_1(\xi_1)$$
$$\xi_1 = w_{12} y_2$$

$$\frac{\partial E}{\partial w_{12}}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$

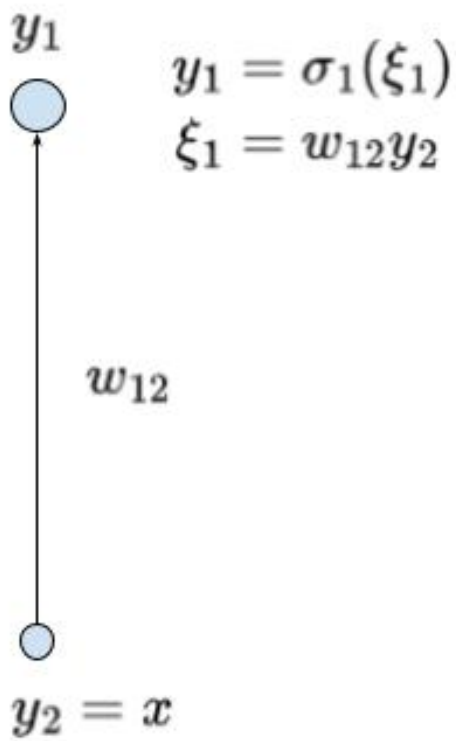



$$y_1 = \sigma_1(\xi_1)$$
$$\xi_1 = w_{12} y_2$$

$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1} \frac{\partial y_1}{\partial w_{12}}$$

$$D = \{(x, d)\}$$

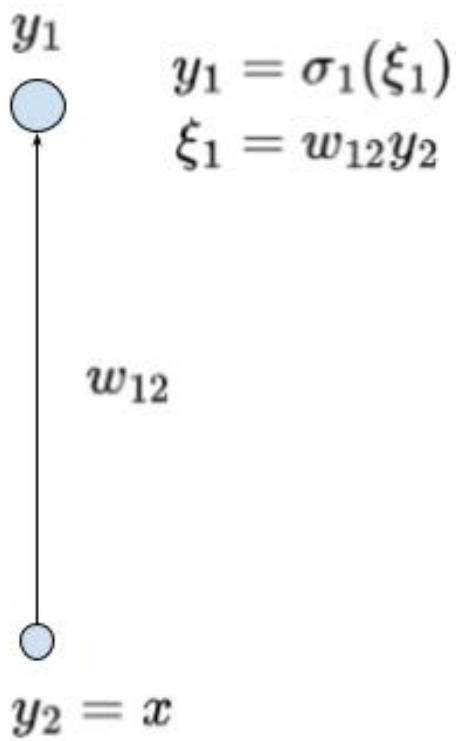
$$E = \frac{1}{2}(y_1 - d)^2$$




$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



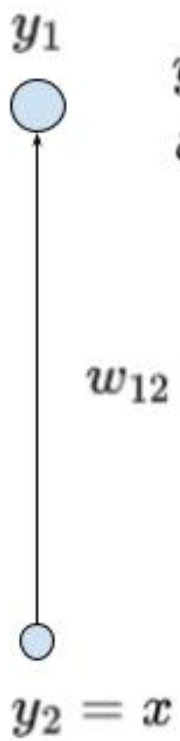

$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}}$$
$$= \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1) y_2$$

Here $\sigma_1'$ is just the plain derivative of $\sigma'$ as a function of a single variable

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$

$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}}$$
$$= \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)y_2$$




$$y_1 = \sigma_1(\xi_1)$$
$$\xi_1 = w_{12}y_2$$

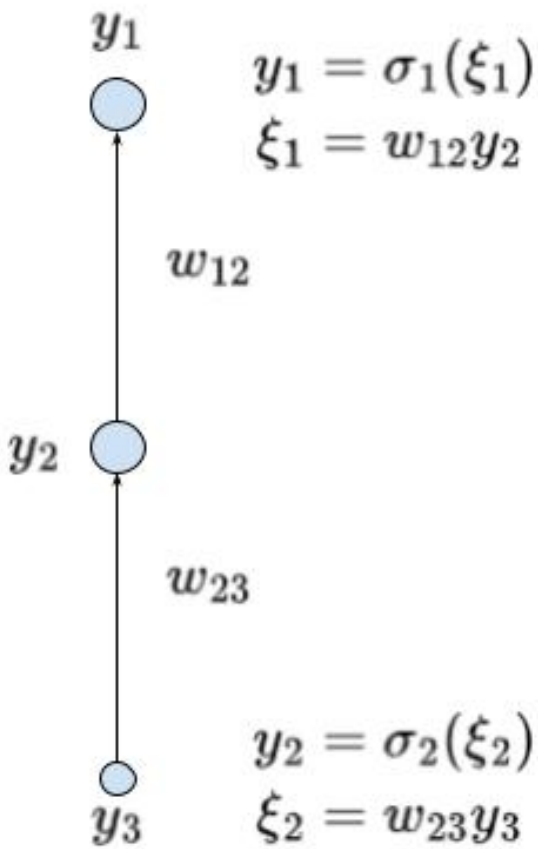
Here $\sigma_1'$ is just the plain derivative of $\sigma'$ as a function of a single variable

$$\frac{\partial E}{\partial y_1} = y_1 - d$$

Note that if $\sigma_1$ is identity, we obtain exactly the gradient from the linear regression method. Considering $\sigma_1$ equal to the logistic sigmoid and $E$ the cross-entropy, we get the logistic regression gradient.

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$




$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1) y_2$$

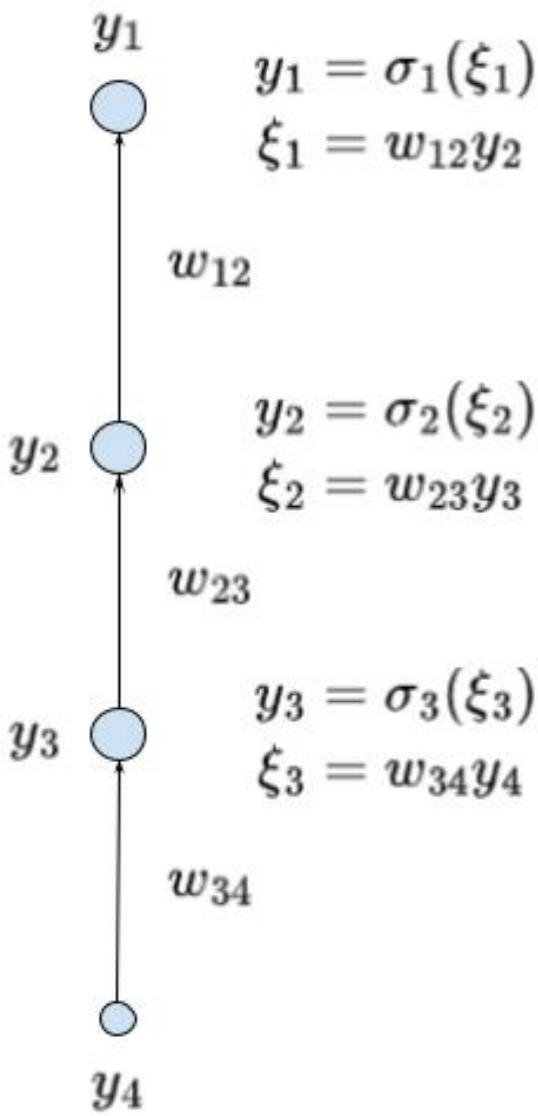
$$\frac{\partial E}{\partial w_{23}} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial w_{23}} = \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2) y_3$$

$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1) w_{12}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$




$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1) y_2$$

$$\frac{\partial E}{\partial w_{23}} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial w_{23}} = \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2) y_3$$

$$\frac{\partial E}{\partial w_{34}} = \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial \xi_3}\frac{\partial \xi_3}{\partial w_{34}} = \frac{\partial E}{\partial y_3}\sigma_3'(\xi_3) y_4$$

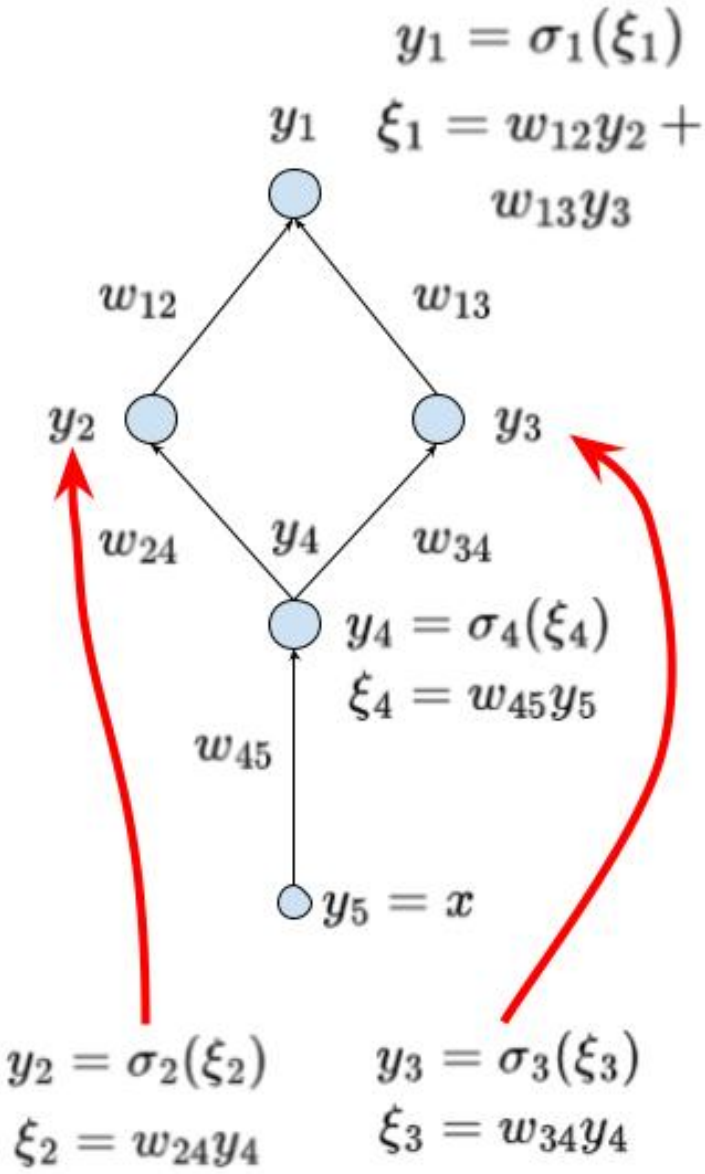
$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1) w_{12}$$

$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial y_3} = \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2) w_{23}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$




$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)y_2$$

$$\frac{\partial E}{\partial w_{13}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_{13}} = \frac{\partial E}{\partial y_1}\sigma_2'(\xi_2)y_3$$

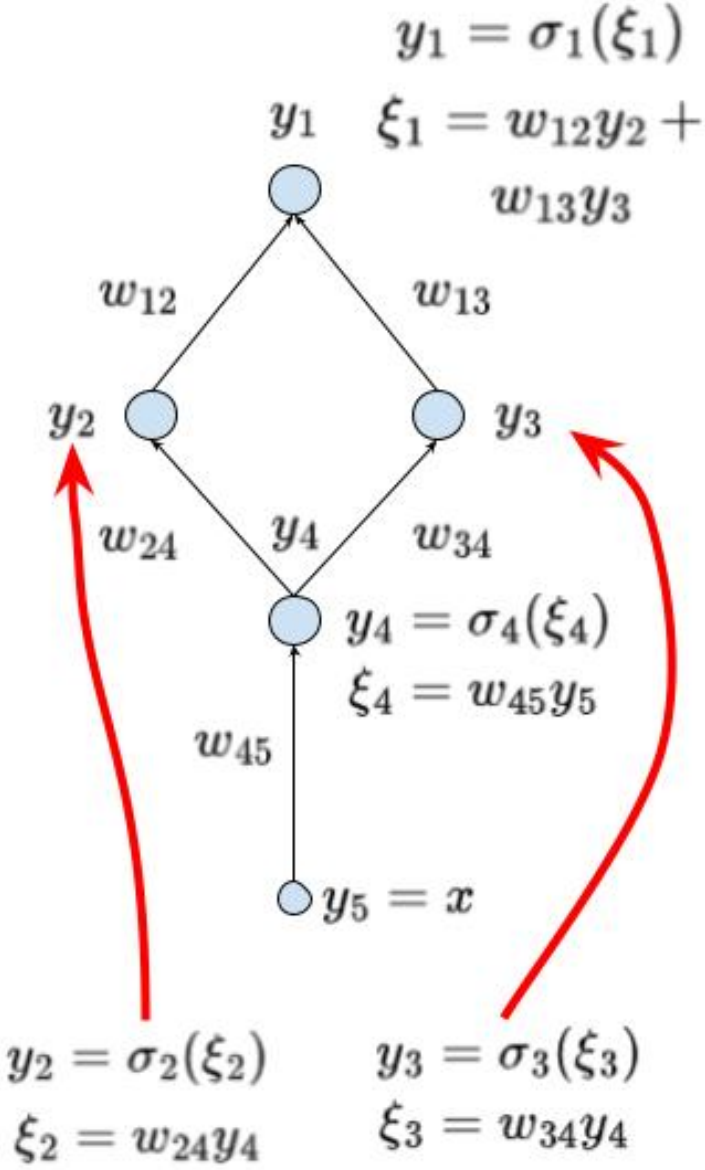
$$\frac{\partial E}{\partial w_{24}} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial w_{24}} = \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2)y_4$$

$$\frac{\partial E}{\partial w_{34}} = \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial \xi_3}\frac{\partial \xi_3}{\partial w_{34}} = \frac{\partial E}{\partial y_3}\sigma_3'(\xi_3)y_4$$

$$\frac{\partial E}{\partial w_{45}} = \frac{\partial E}{\partial y_4}\frac{\partial y_4}{\partial \xi_4}\frac{\partial \xi_3}{\partial w_{45}} = \frac{\partial E}{\partial y_4}\sigma_4'(\xi_4)y_5$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



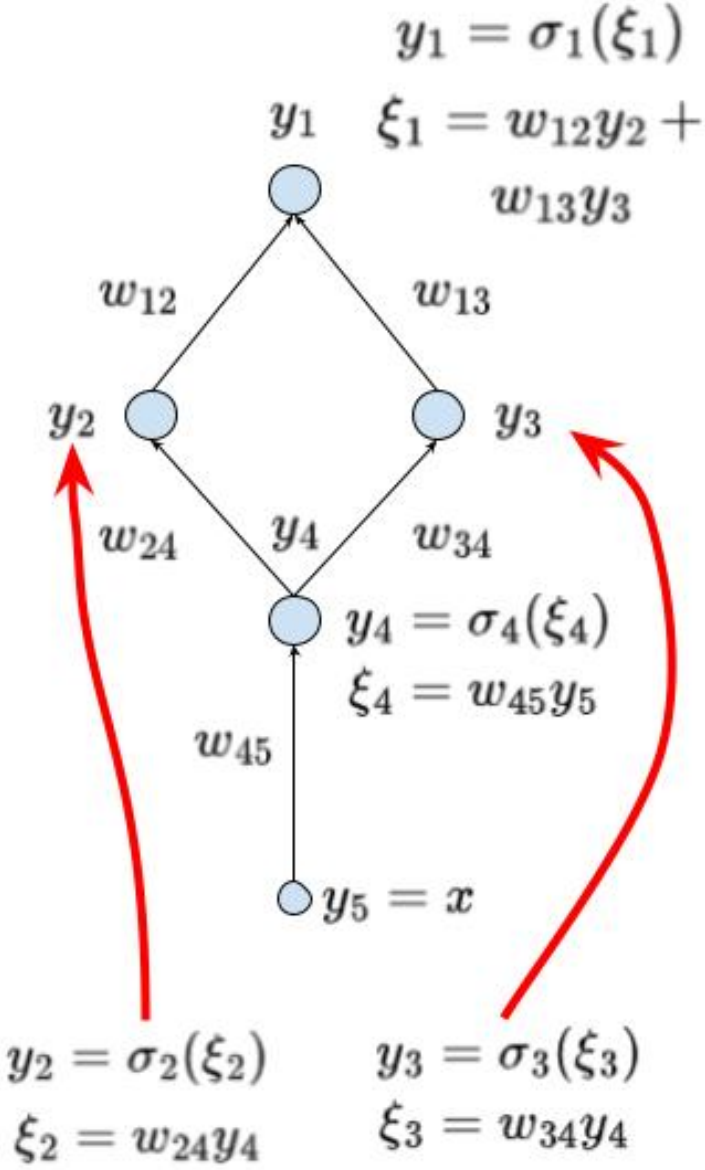

$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_3} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{13}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



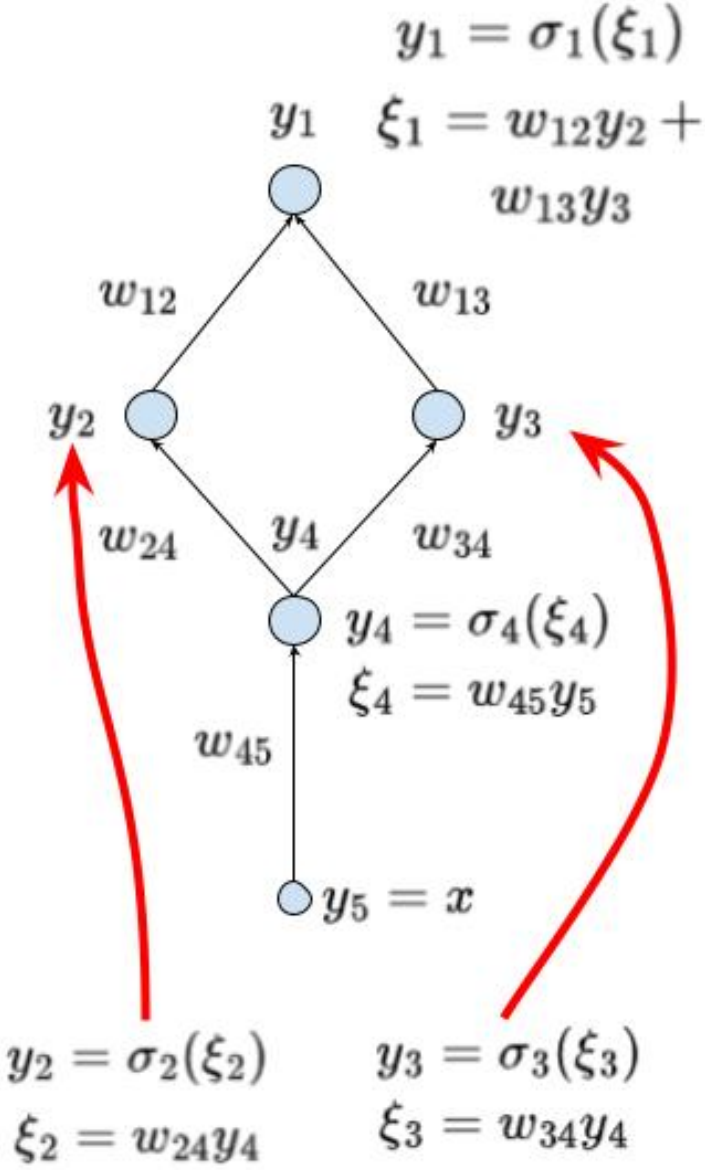

$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_3} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{13}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



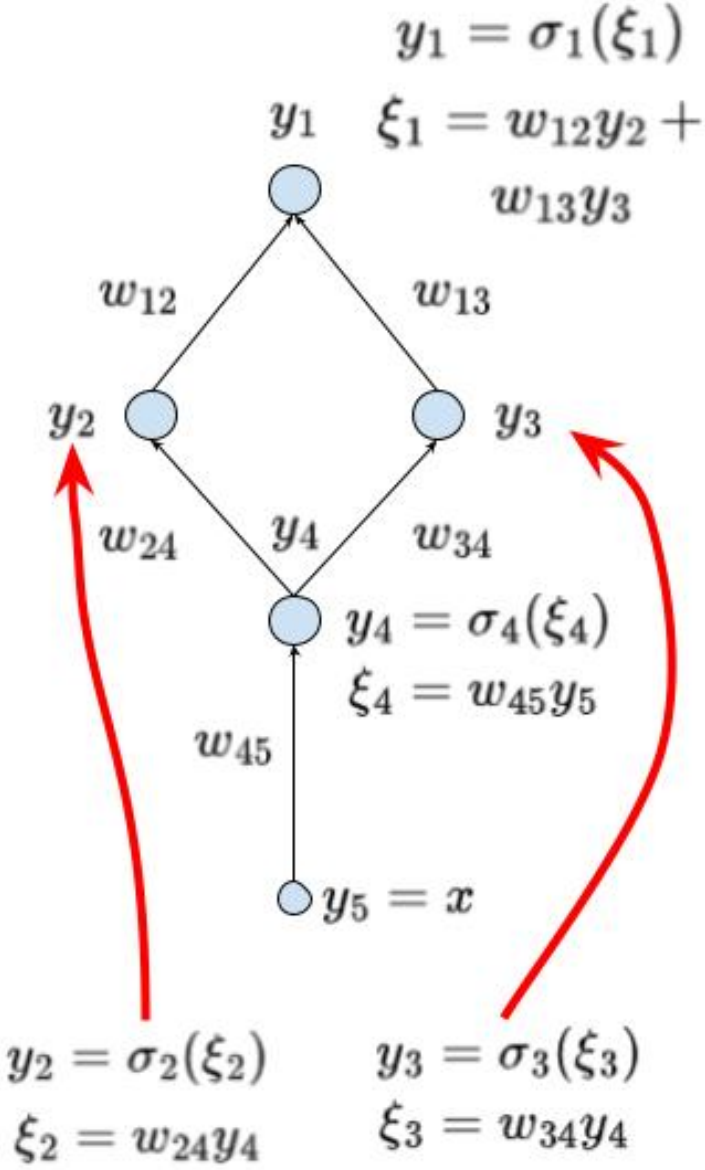

$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_3} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{13}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$



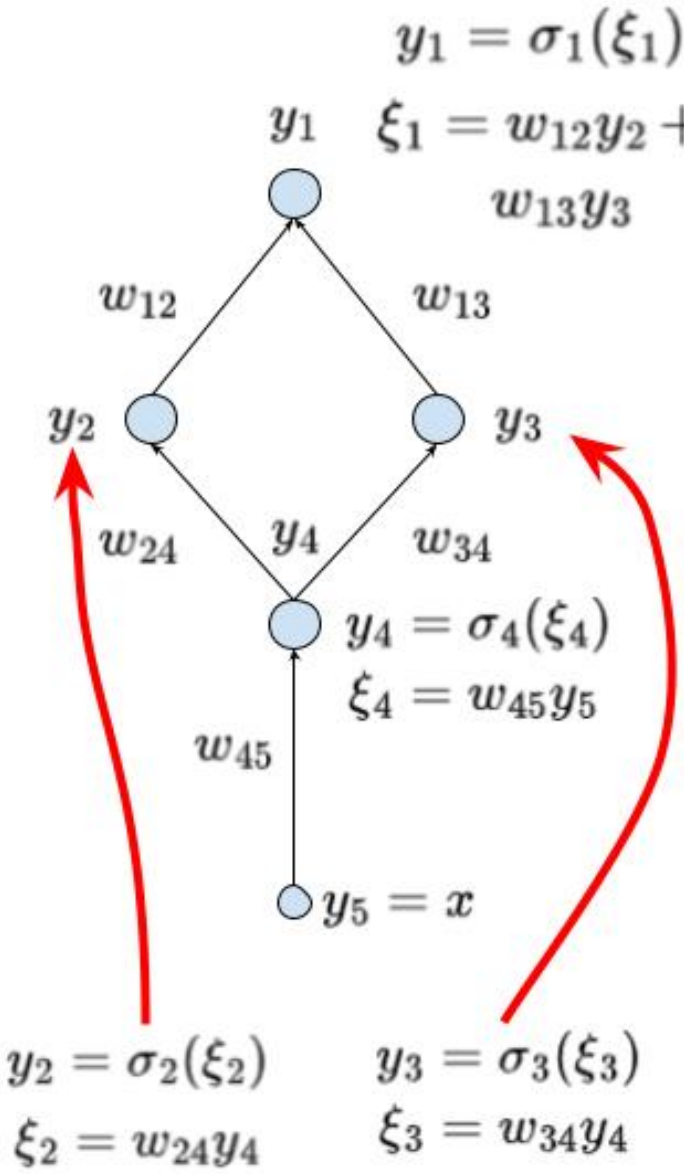

$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_3} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{13}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$




$$\frac{\partial E}{\partial y_1} = y_1 - d$$

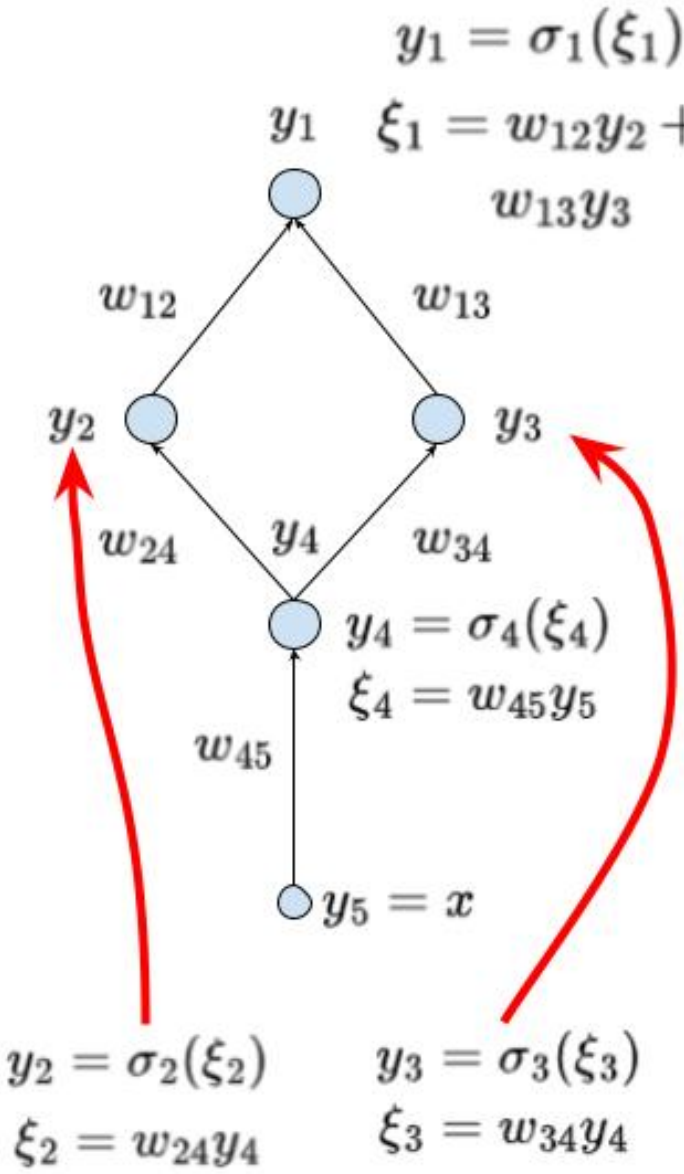
$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_3} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{13}$$

$$\frac{\partial E}{\partial y_4}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$




$$\frac{\partial E}{\partial y_1} = y_1 - d$$

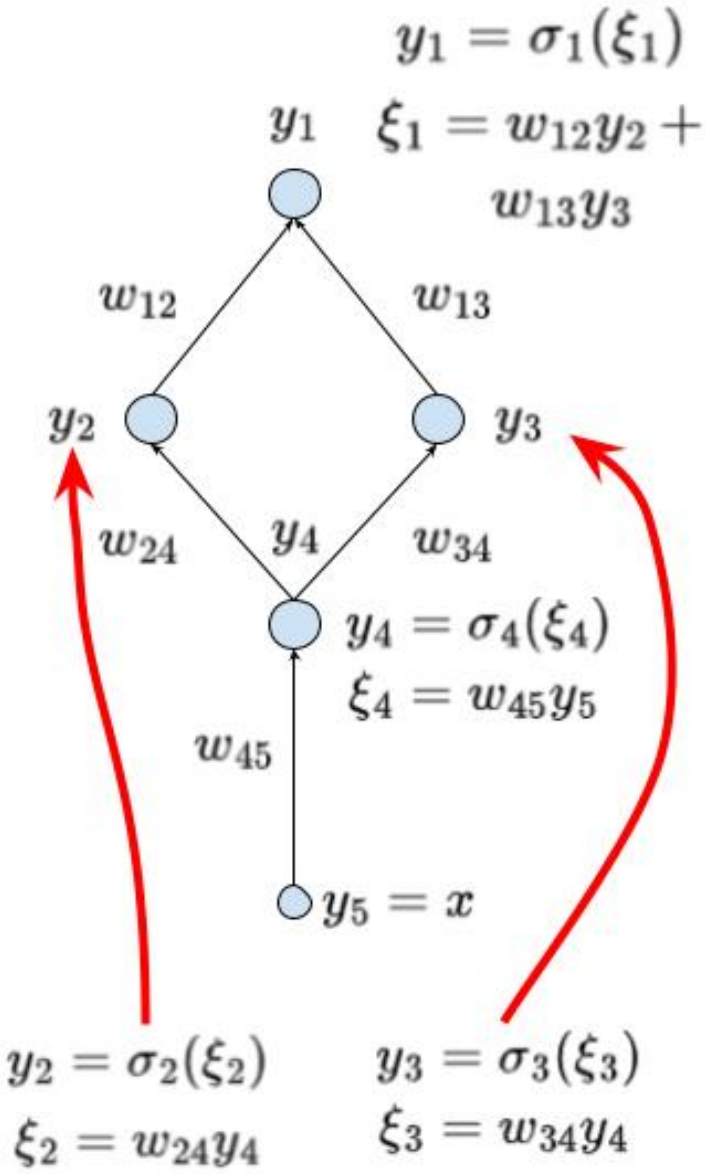
$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_3} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{13}$$

$$\frac{\partial E}{\partial y_4} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial y_4} + \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial y_4}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$




$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

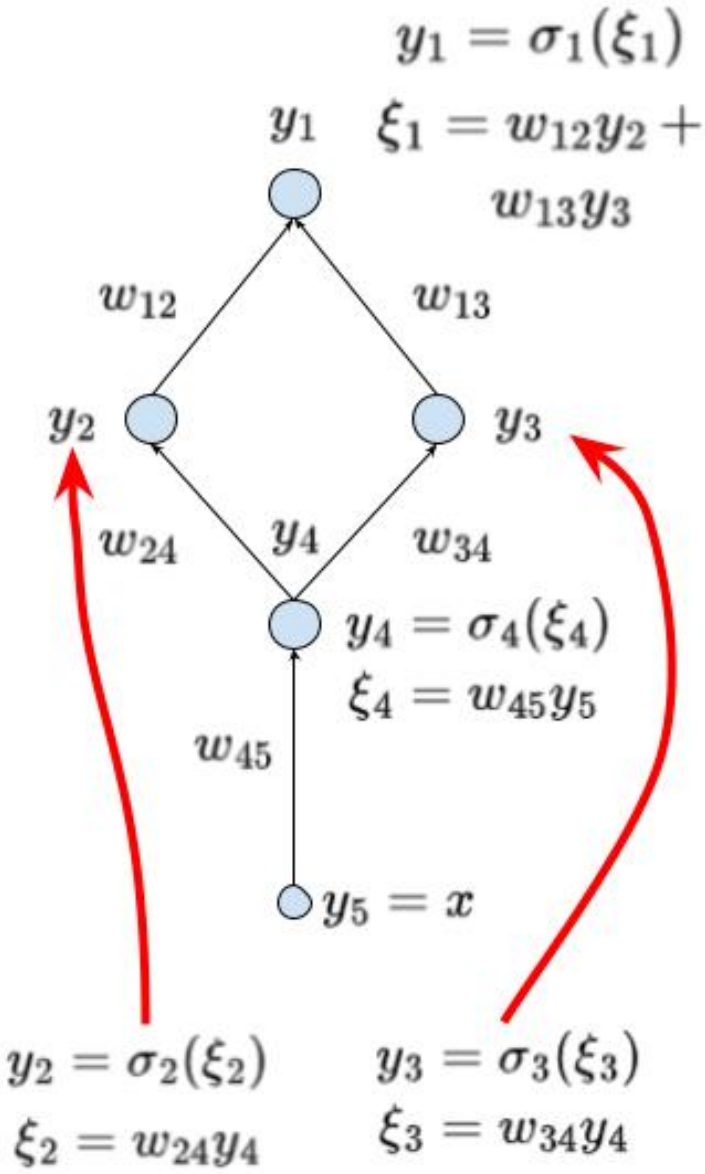
$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_3} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{13}$$

$$\frac{\partial E}{\partial y_4} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial y_4} + \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial y_4}$$

$$= \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial y_4} + \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial \xi_3}\frac{\partial \xi_3}{\partial y_4}$$

$$D = \{(x, d)\}$$

$$E = \frac{1}{2}(y_1 - d)^2$$




$$\frac{\partial E}{\partial y_1} = y_1 - d$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_2} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}$$

$$\frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial y_3} = \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{13}$$

$$\begin{aligned}\frac{\partial E}{\partial y_4} &= \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial y_4} + \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial y_4} \\ &= \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial y_4} + \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial \xi_3}\frac{\partial \xi_3}{\partial y_4} \\ &= \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2)w_{24} + \frac{\partial E}{\partial y_3}\sigma_3'(\xi_3)w_{34}\end{aligned}$$

# MLP – Gradient Computation

Under our simplifying assumptions $D = \{(x, d)\}$ and $E = (y_1 - d)^2$ the gradient computaton proceeds as follows:

Applying the chain rule, we obtain

$$\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial y_j} \cdot \sigma_j'(\xi_j) \cdot y_i$$

where (after more applications of the chain rule)

$$\frac{\partial E_k}{\partial y_1} = y_1 - d$$

Keep in mind that 1 is the only output neuron which means that $y_1$ is the value of the network.

$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot w_{rj} \qquad \text{for } j \in Z \setminus (Y \cup X)$$

Here $y_r = y[\vec{w}](x)$ where $\vec{w}$ are the current weights and $x$ is the example input.

# MLP – Gradient Computation – General!

Let us drop our simplifying assumptions!

- Given a set $D$ of training examples:

$$D = \left\{ \left(\vec{x}_k, \vec{d}_k\right) \quad | \quad k = 1, \ldots, p \right\}$$

  Here $\vec{x}_k \in \mathbb{R}^{|X|}$ and $\vec{d}_k \in \mathbb{R}^{|Y|}$. We write $d_{kj}$ to denote the value in $\vec{d}_k$ corresponding to the output neuron $j$.

- **Error Function:** $E(\vec{w})$ where $\vec{w}$ is a vector of all weights in the network. The choice of $E$ depends on the solved task (classification vs regression, etc.).
  **Example (Squared error):**

$$E(\vec{w}) = \sum_{k=1}^{p} E_k(\vec{w})$$

  where

$$E_k(\vec{w}) = \frac{1}{2} \sum_{j \in Y} (y_j[\vec{w}](\vec{x}_k) - d_{kj})^2$$

# MLP – Gradient Computation

For every weight $w_{ji}$ we have (obviously)

$$\frac{\partial E}{\partial w_{ji}} = \sum_{k=1}^{p} \frac{\partial E_k}{\partial w_{ji}}$$

So now it suffices to compute $\frac{\partial E_k}{\partial w_{ji}}$, that is the error for a fixed training example $(\vec{x}_k, \vec{d}_k)$.

Applying the chain rule, we obtain

$$\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial y_j} \cdot \sigma_j'(\xi_j) \cdot y_i$$

where (more applications of the chain rule)

$\frac{\partial E_k}{\partial y_j}$ is computed directly for the output neurons $j \in Y$

$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot w_{rj} \qquad \text{for } j \in Z \setminus (Y \cup X)$$

(Here $y_r = y[\vec{w}](\vec{x}_k)$ where $\vec{w}$ are the current weights and $\vec{x}_k$ is the input of the $k$-th training example.)

# MLP – Backpropagation

**Input:** A training set $D = \left\{ \left(\vec{x}_k, \vec{d}_k\right) \quad | \quad k = 1, \ldots, p \right\}$ and the current vector of weights $\vec{w}$.

Note that the backprop. is repeated in every iteration of the gradient descent!

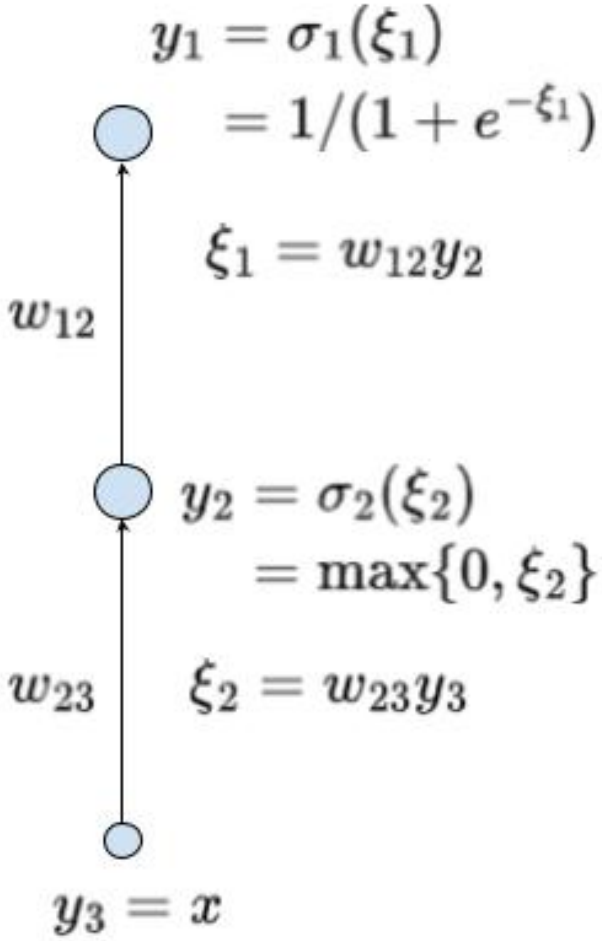
- Evaluate all values $y_i$ of neurons using the standard bottom-up procedure with the input $\vec{x}_k$.
- For every training example $(\vec{x}_k, \vec{d}_k)$ compute $\frac{\partial E_k}{\partial y_j}$ using *backpropagation* through layers top-down :
  - For all $j \in Y$ compute $\frac{\partial E_k}{\partial y_j}$ by taking the derivative of the error.
    e.g., in the case of the squared error, we have $\frac{\partial E_k}{\partial y_j} = y_j - d_{kj}$.
  - In the layer $\ell$, assuming that $\frac{\partial E_k}{\partial y_r}$ has been computed for all neurons $r$ in the layer $\ell + 1$, compute

$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot w_{rj}$$

    for all $j$ from the $\ell$-th layer. Here $\sigma_r'$ is the derivative of $\sigma_r$.
  - Put $\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial y_j} \cdot \sigma_j'(\xi_j) \cdot y_i$

**Output:** $\frac{\partial E}{\partial w_{ji}} = \sum_{k=1}^{p} \frac{\partial E_k}{\partial w_{ji}}$.

# MLP Learning Example




Training set:

$$D = \{(x, d)\} = \{(1, 1)\}$$

That is

$$y_3 = x = 1$$
$$d = 1$$

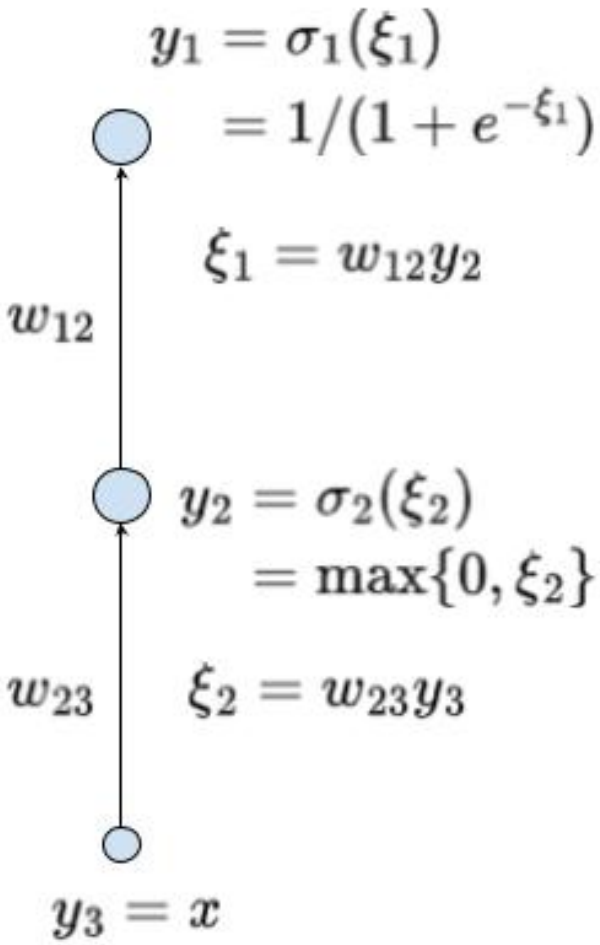
Error cross-entropy:

$$E(\vec{w}) = -(d \log(y_1) + (1 - d) \log(1 - y_1))$$
$$= -\log(y_1)$$

Assume the initial weight vector
$\vec{w}^{(0)} = (w_{12}^{(0)}, w_{23}^{(0)}) = (\frac{1}{4}, 2)$.

Consider the learning rate $\varepsilon = 0.5$.

# MLP Learning Example – Gradient Descent



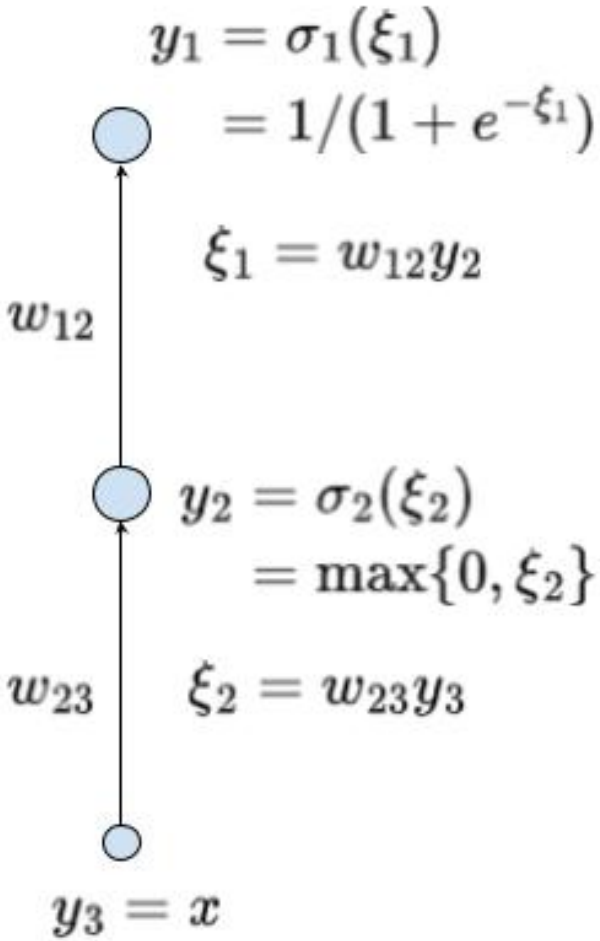

To make the gradient descent step:

$$w_{12}^{(1)} = w_{12}^{(0)} - \varepsilon \frac{\partial E}{\partial w_{12}}(\vec{w}^{(0)})$$

$$w_{23}^{(1)} = w_{23}^{(0)} - \varepsilon \frac{\partial E}{\partial w_{23}}(\vec{w}^{(0)})$$

we need to compute the partial derivatives $\frac{\partial E}{\partial w_{12}}$ and $\frac{\partial E}{\partial w_{23}}$.

# MLP Learning Example – Forward Pass

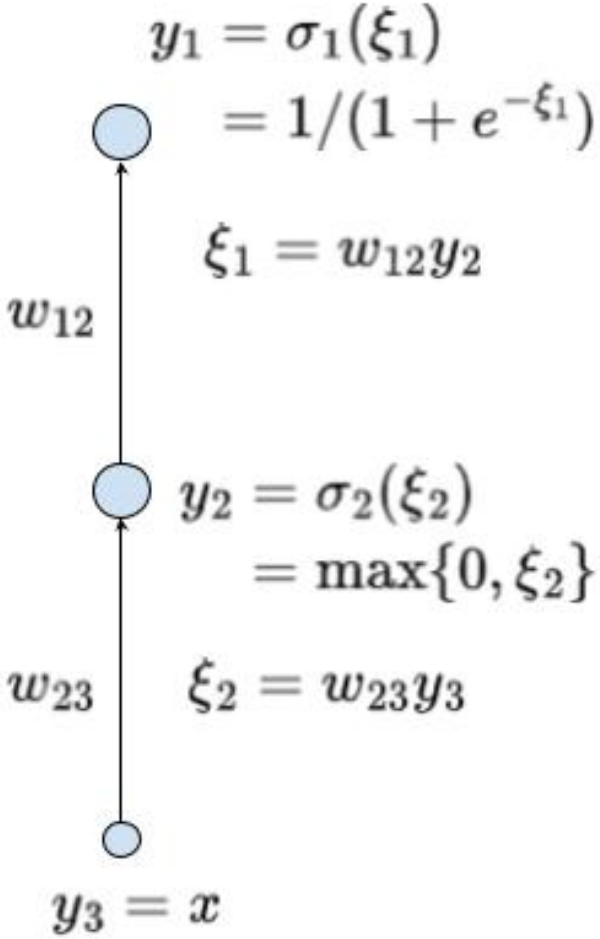



We have $x = 1, w_{12}^{(0)} = 1/4, w_{23}^{(0)} = 2$

First, compute the **forward pass**

$$y_3 = x = 1$$
$$\xi_2 = w_{23}^{(0)} y_3 = 2y_3 = 2$$
$$y_2 = \max\{0, \xi_2\} = 2$$
$$\xi_1 = w_{12}^{(0)} y_2 = \frac{1}{4}2 = \frac{1}{2}$$
$$y_1 = 1/(1+e^{-\xi_1}) = 1/(1+e^{-(1/2)})$$
$$= 0.6225$$

# MLP Learning Example – Backward Pass




We have $w_{12}^{(0)} = 1/4, w_{23}^{(0)} = 2, y_1 = 0.6225, y_2 = 2, y_3 = 1$.

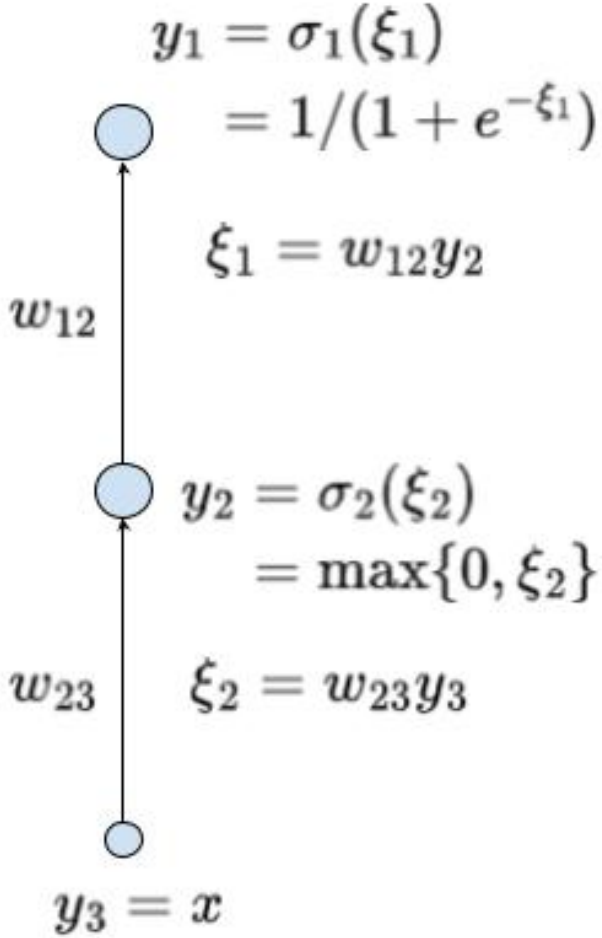
Proceed with the backward pass:

$$\frac{\partial E}{\partial y_1} = \frac{\partial(-\log(y_1))}{\partial y_1} = -\frac{1}{y_1} = -1.6065$$

Since $\sigma_1' = \sigma_1(1-\sigma_1)$

$$\begin{aligned}
\frac{\partial E}{\partial y_2} &= \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1)w_{12}^{(0)} \\
&= \frac{\partial E}{\partial y_1}\sigma_1(\xi_1)(1-\sigma_1(\xi_1))w_{12}^{(0)} \\
&= \frac{\partial E}{\partial y_1}y_1(1-y_1)w_{12}^{(0)} \\
&= -1.6065 \cdot 0.6225 \cdot 0.3775 \cdot (1/4) \\
&= -0.09438
\end{aligned}$$

# MLP Learning Example – The Gradient




We have
$w_{12}^{(0)} = 1/4, w_{23}^{(0)} = 2, y_1 = 0.6225, y_2 = 2, y_3 = 1, \frac{\partial E}{\partial y_1} = -1.6065, \frac{\partial E}{\partial y_2} = -0.09438.$

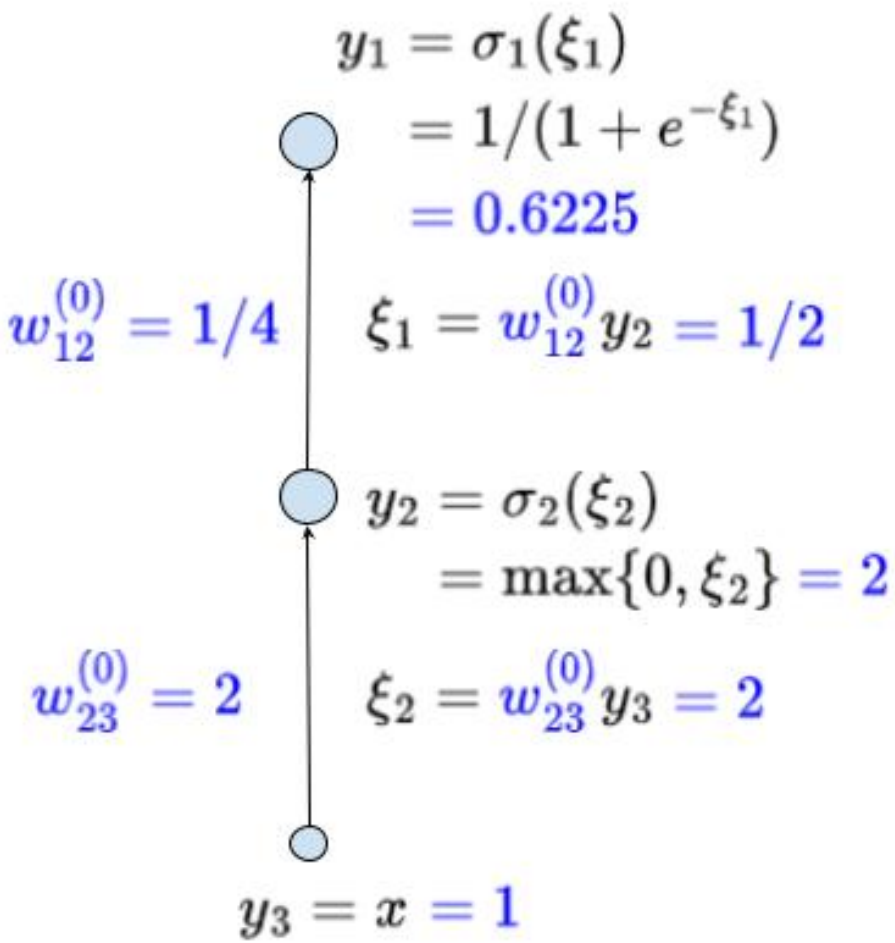
Compute derivatives of $E$ w.r.t. weights:

$$\begin{aligned}\frac{\partial E}{\partial w_{12}} &= \frac{\partial E}{\partial y_1}\sigma_1'(\xi_1) y_2 \\ &= \frac{\partial E}{\partial y_1} y_1(1-y_1) y_2 \\ &= -0.755\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_{23}} &= \frac{\partial E}{\partial y_2}\sigma_2'(\xi_2) y_3 \\ &= \frac{\partial E}{\partial y_2} y_3 \\ &= -0.09438\end{aligned}$$

# Backpropagation – Example – Summary

**Forward pass (bottom up)**



# Backpropagation – Example – Summary

**Forward pass (bottom up)**

$$y_1 = \sigma_1(\xi_1) = 1/(1+e^{-\xi_1}) = 0.6225$$

$$w_{12}^{(0)} = 1/4 \qquad \xi_1 = w_{12}^{(0)} y_2 = 1/2$$

$$y_2 = \sigma_2(\xi_2) = \max\{0, \xi_2\} = 2$$

$$w_{23}^{(0)} = 2 \qquad \xi_2 = w_{23}^{(0)} y_3 = 2$$

$$y_3 = x = 1$$

**Backward pass (top down)**

$$\frac{\partial E}{\partial y_1} = 1/y_1 = -1.6065$$

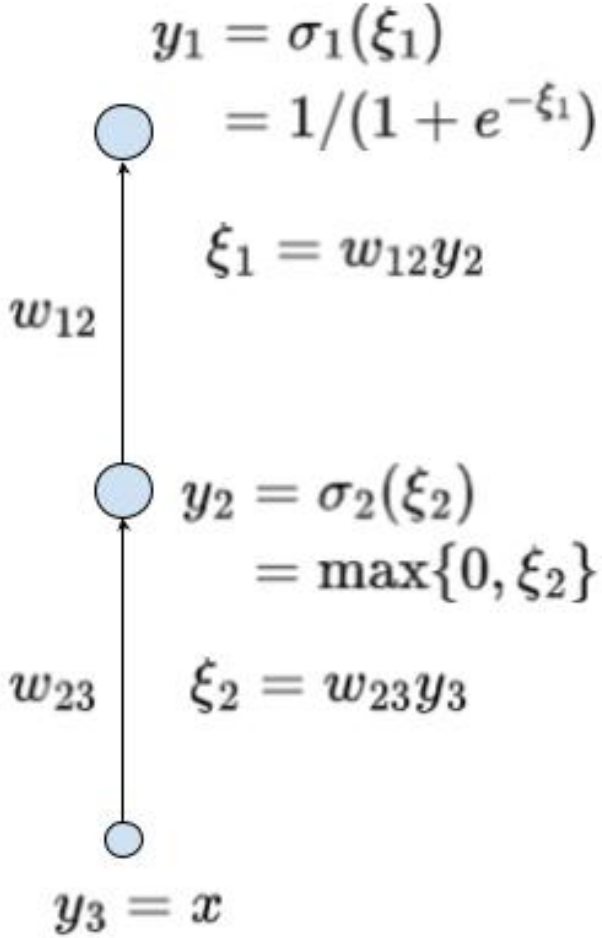
$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial y_1} y_1(1-y_1) y_2 = -0.755$$

$$\frac{\partial E}{\partial y_2} = \frac{\partial E}{\partial y_1} y_1(1-y_1) w_{12}^{(0)} = -0.09438$$

$$\frac{\partial E}{\partial w_{23}} = \frac{\partial E}{\partial y_2} 1 y_3 = -0.09438$$

Note that **WE HAVE NOT YET CHANGED ANY WEIGHTS!**

# MLP Learning Example – Gradient Descent Step




So **ONLY NOW** we can make the **learning step** and change the weights:

$$
\begin{aligned}
w_{12}^{(1)} &= w_{12}^{(0)} - \varepsilon \frac{\partial E}{\partial w_{12}}(\vec{w}^{(0)}) \\
&= \frac{1}{4} - 0.5 \cdot (-0.755) \\
&= 0.627
\end{aligned}
$$

$$
\begin{aligned}
w_{23}^{(1)} &= w_{23}^{(0)} - \varepsilon \frac{\partial E}{\partial w_{23}}(\vec{w}^{(0)}) \\
&= 2 - 0.5 \cdot (-0.09438) \\
&= 2.047
\end{aligned}
$$

We have made just a **single** gradient descent step!

# MLP Training Summary

- MLP are trained using the **gradient descent** algorithm

  In practice, modifications of GD are typically used, but most of them have strong roots in GD.
- The gradients are computed using the backpropagation algorithm

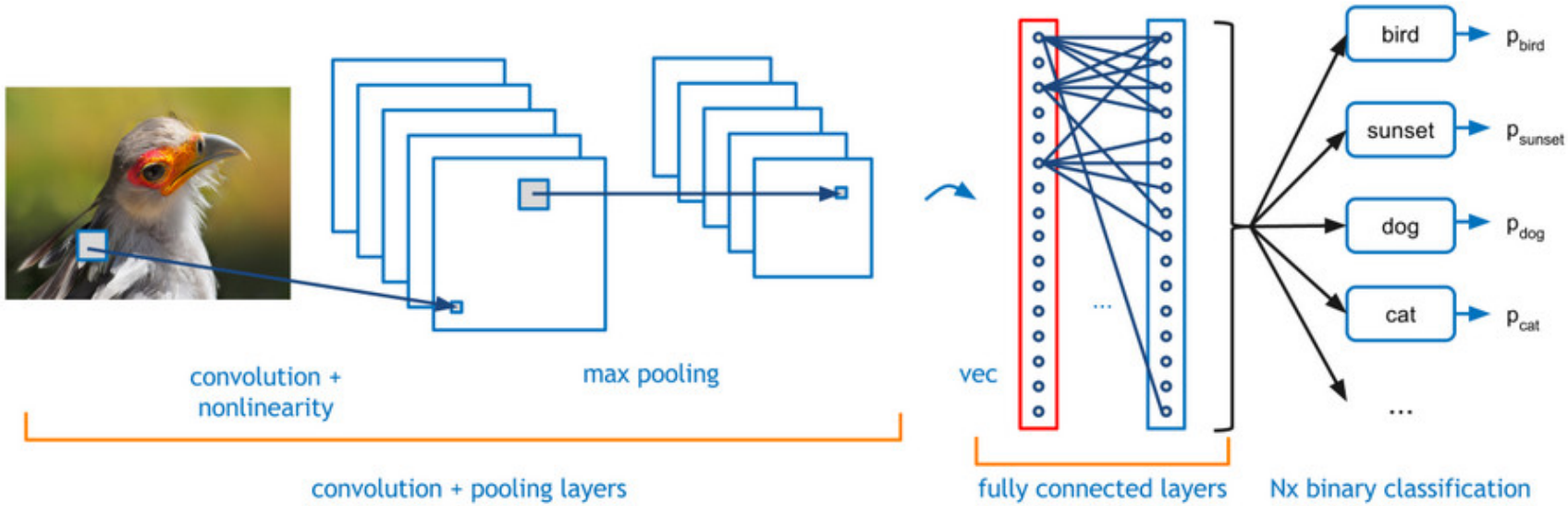
  the backpropagation is a universal method for automatic differentiation, surpassing any use in neural networks.
- Training of neural networks in practice is tricky due to many reasons comprising in particular
  - highly complex non-linear shape of the error function,
  - tendency to overfit very quickly,
  - black-box nature, hard to see what the network does,
  - huge hype around deep learning (which many people confuse with AI) results in high expectations even in cases where no (learning) algorithm may solve the given problem!

**An advice:** Always concentrate the main effort on the solved problem formulation and, afterward, on the data you have at your disposal (and honestly separate the Test set right at the beginning).

# Deep Learning

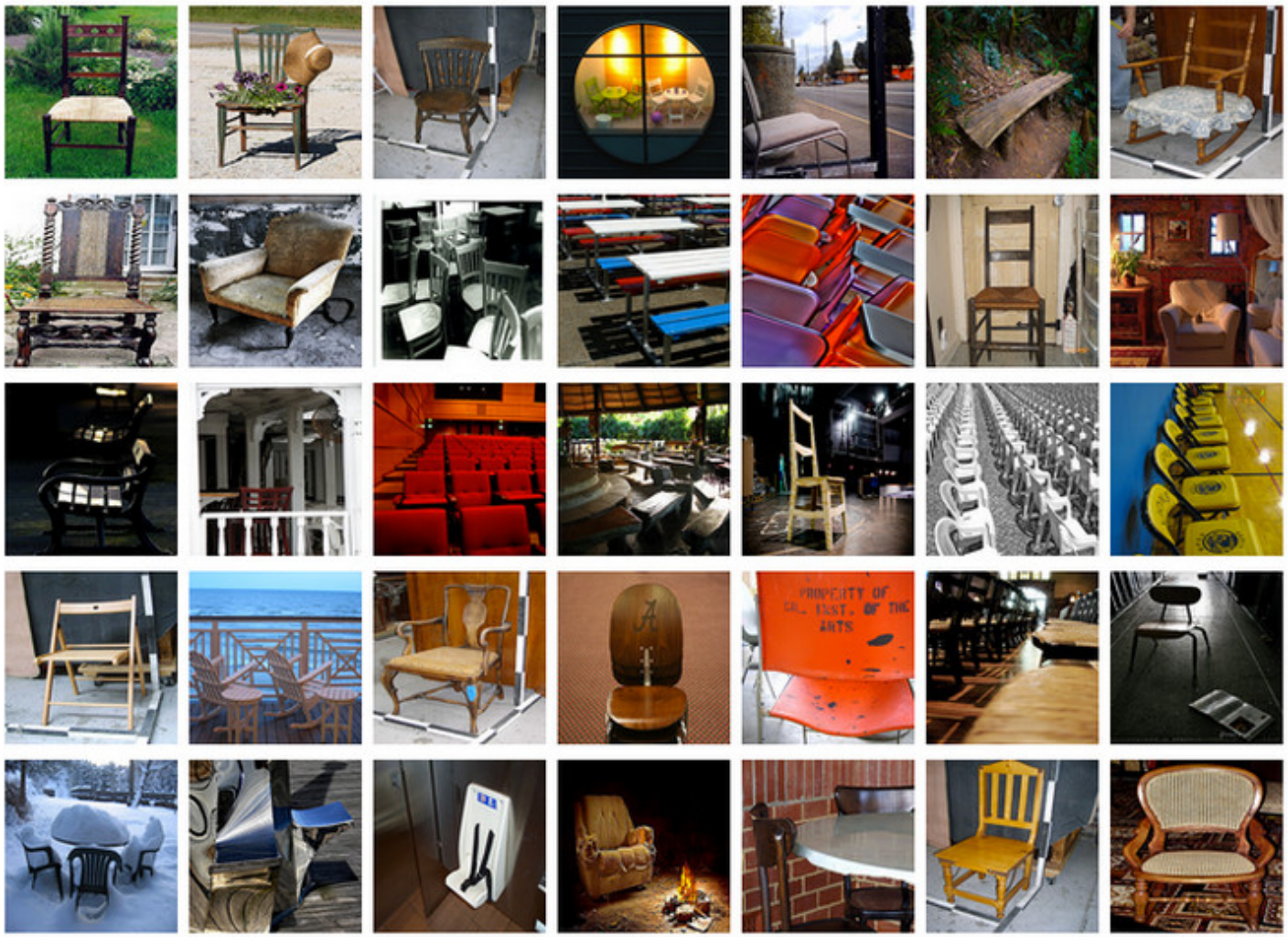
- Cybenko's theorem shows that two-layer networks are omnipotent – such results nearly killed NN when support vector machines were found to be easier to train in 00's.
- Later, it has been shown (experimentally) that deep networks (with many layers) have better representational properties.
- ... but how to train them? The gradient descent suffers from a so-called vanishing gradient; intuitively, updates of weights in lower layers are *very* slow.
- In 2006, a solution was found by Hinton et al.:
  - Use *unsupervised* methods to initialize the weights layer by layer to capture important data features.
    More precisely: The lowest hidden layer learns patterns in data, the second lowest learns patterns in data transformed through the first layer, and so on.
  - Then use a supervised learning algorithm to only *fine tune* the weights to the desired input-output behavior.
- ... but the actual revolution started with convolutional networks trained on several GPUs.

# Convolutional network



# ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)

ImageNet database (16,000,000 color images, 20,000 categories)



# ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)

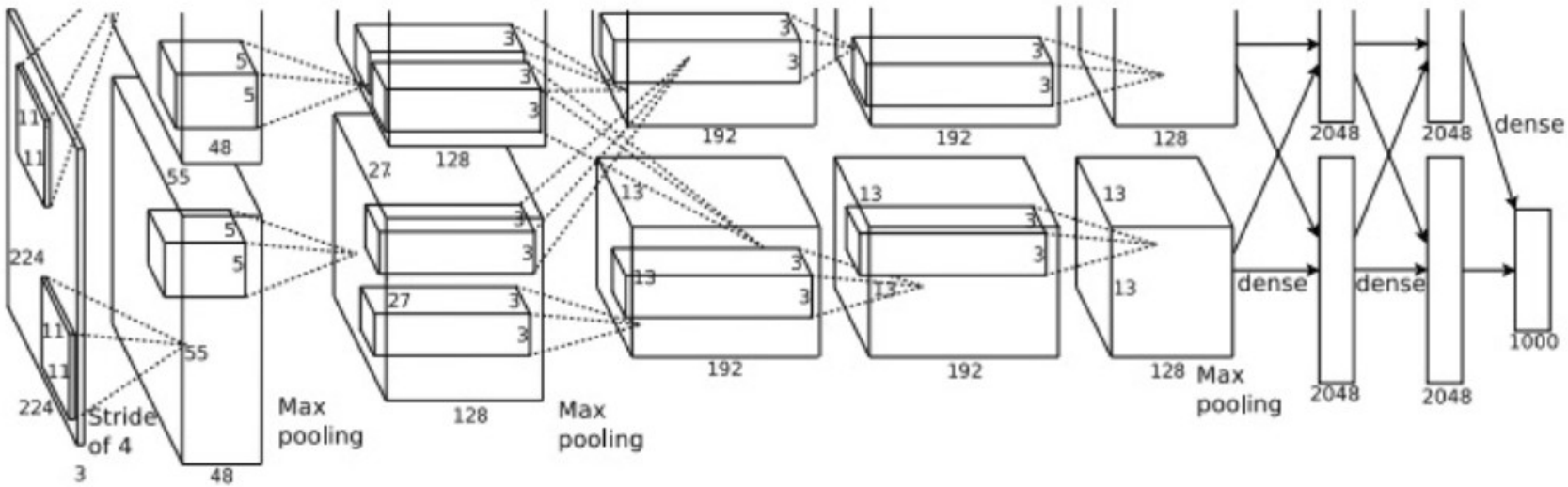
Competition in classification over a subset of images from ImageNet.

In 2012, training saw 1,200,000 images and 1000 categories. Validation set 50,000, Test set 150,000.

Many images contain several objects $\rightarrow$ typical rule is top-5 highest probability assigned by the net.

# KSH net

ImageNet classification with deep convolutional neural networks, by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton (2012).




Trained on two GPUs (NVIDIA GeForce GTX 580)

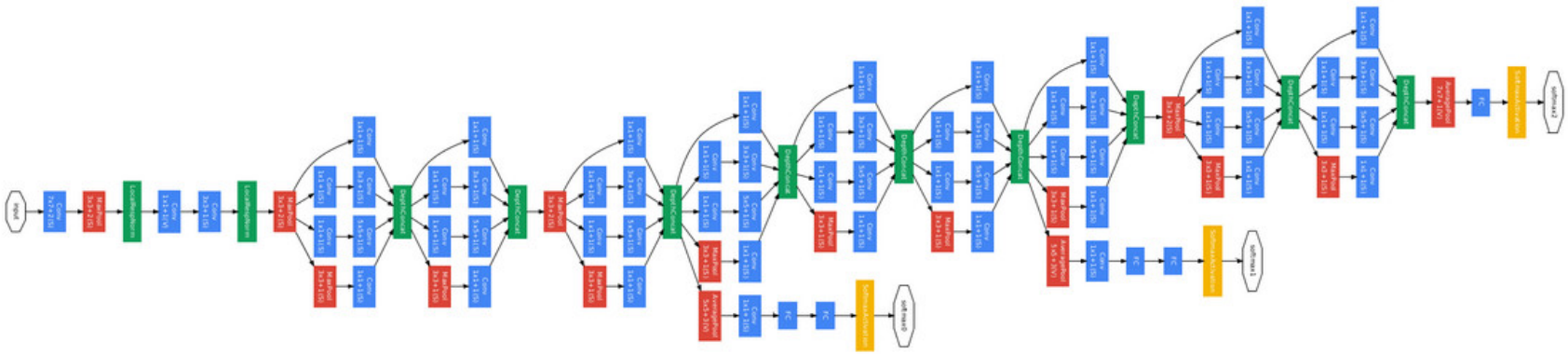
Results:

- Accuracy 84.7% in top-5 (second best alg. at the time: 73.8%)
- 63.3% in "perfect" classification (top-1)

# ILSVRC 2014

The same set of images as in 2012, top-5 criterium.

GoogLeNet: deep convolutional net, 22 layers



Results:

- 93.33% in top-5

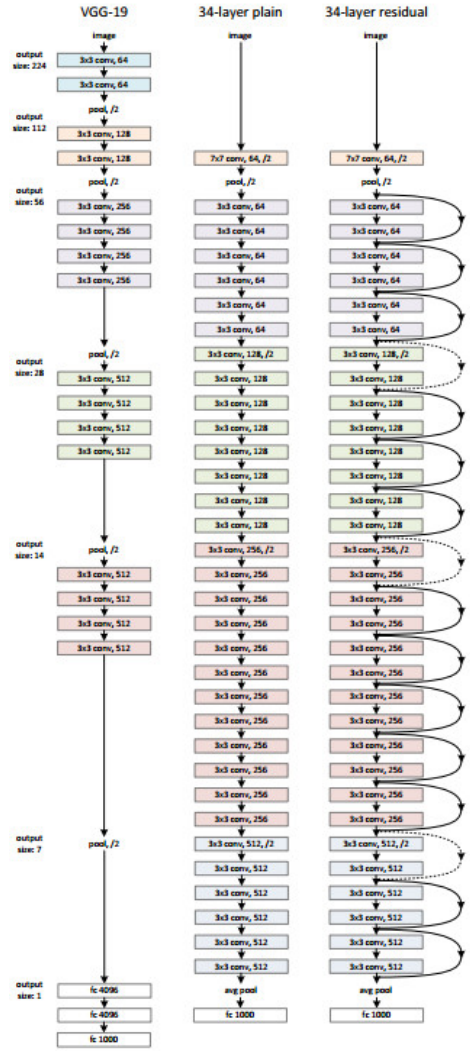
Superhuman power?

# Superhuman GoogLeNet?!

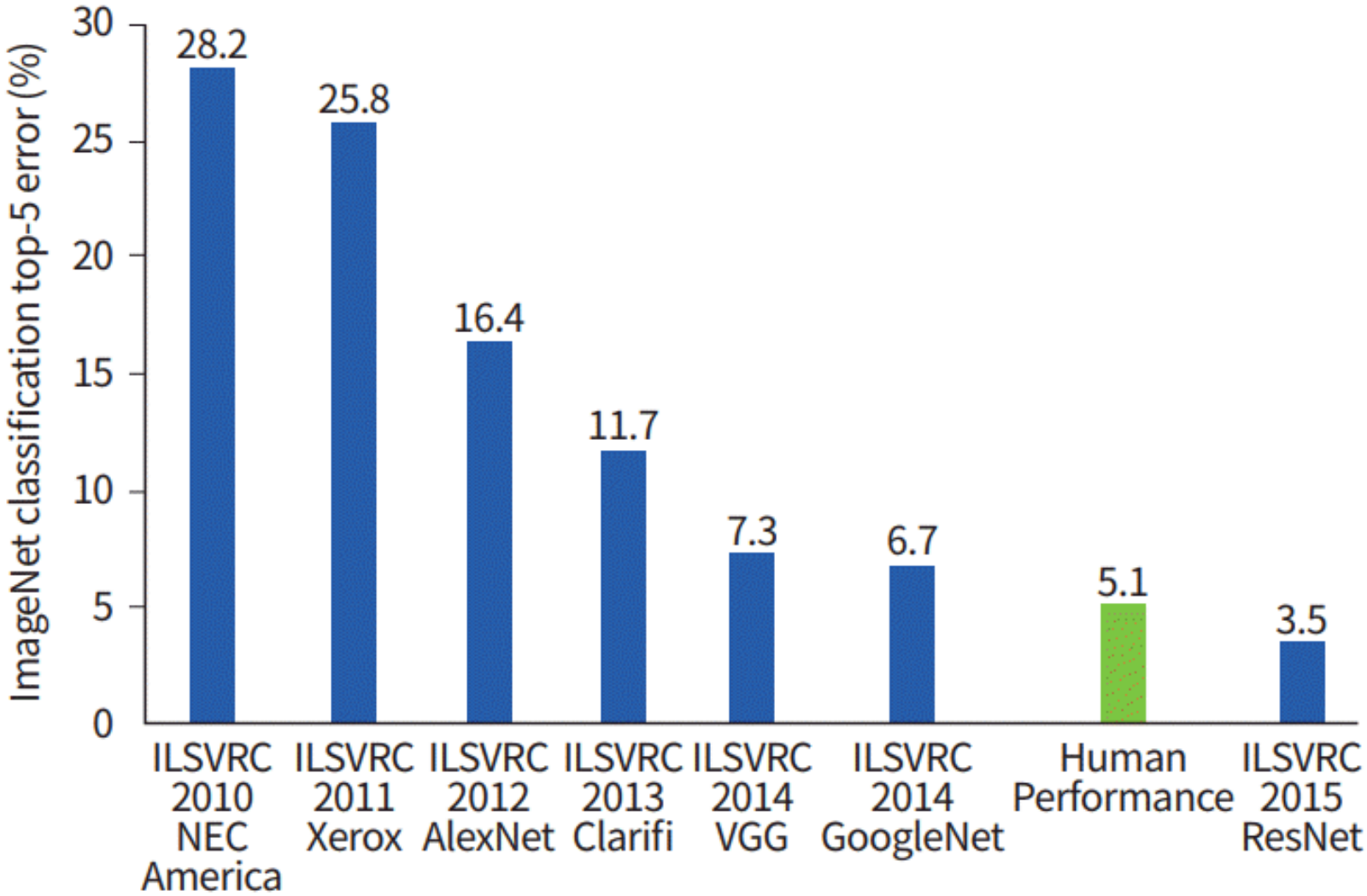
Andrej Karpathy: ...the task of labeling images with 5 out of 1000 categories quickly turned out to be highly challenging, even for some friends in the lab who have been working on ILSVRC and its classes for a while. First, we thought we would put it up on [Amazon Mechanical Turk]. Then, we thought we could recruit paid undergrads. Then, I organized a labeling party of intense labeling effort only among the (expert labelers) in our lab. Then, I developed a modified interface that used GoogLeNet predictions to prune the number of categories from 1000 to only about 100. It was still too hard - people kept missing categories and getting up to ranges of 13-15% error rates. In the end, I realized that to get anywhere competitively close to GoogLeNet, it would be most efficient if I sat down and went through the painfully long training process and the subsequent careful annotation process myself. The labeling happened at a rate of about 1 per minute, but this decreased over time... Some images are easily recognized, while some pictures (such as those of fine-grained breeds of dogs, birds, or monkeys) can require multiple minutes of concentrated effort. I became very good at identifying breeds of dogs... Based on the sample of images I worked on, the GoogLeNet classification error turned out to be 6.8%... In the end, my error turned out to be 5.1%

# ILSVRC 2015

- Microsoft network ResNet: 152 layers, complex architecture
- Trained on 8 GPUs
- 96.43% **accuracy** in top-5



# ILSVRC


ImageNet classification top-5 error (%)
30
25
20
15
10
5
0
28.2
25.8
16.4
11.7
7.3
6.7
5.1
3.5
ILSVRC 2010 NEC America
ILSVRC 2011 Xerox
ILSVRC 2012 AlexNet
ILSVRC 2013 Clarifi
ILSVRC 2014 VGG
ILSVRC 2014 GoogleNet
Human Performance
ILSVRC 2015 ResNet

# ILSVRC 2016

Trumps-Soushen (The Third Research Institute of Ministry of Public Security)

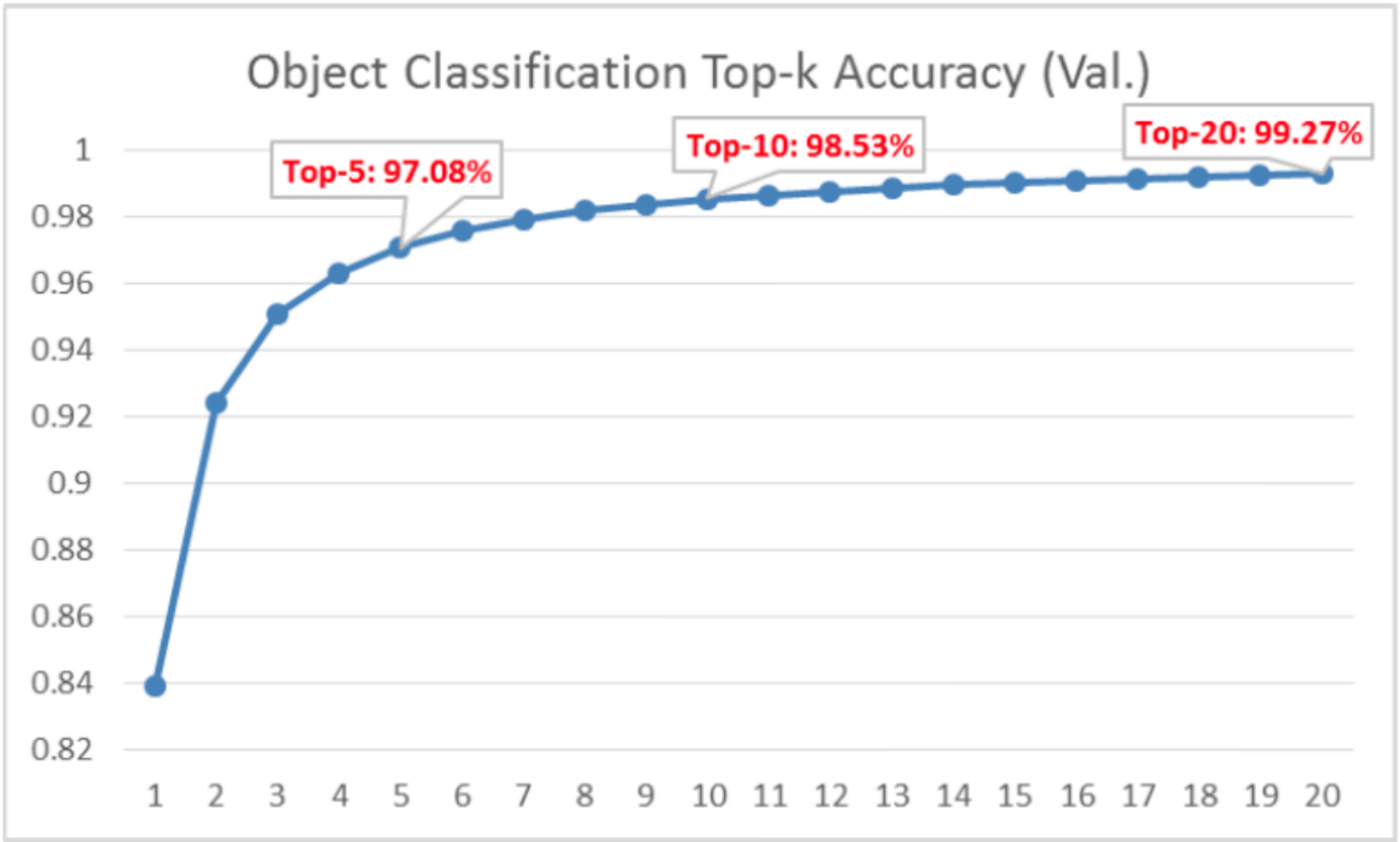
There is no new innovative technology or novelty by Trimps-Soushen.

Ensemble of the pre-trained models from previous years.

Each model is strong at classifying some categories but weak at categorizing others.

Test error: 2.99%

# Top-k accuracy analyzed




https://towardsdatascience.com/review-trimps-soushen-winner-in-ilsvrc-2016-image-classification-dfbc423111dd

# Top-20 typical errors

Out of 1458 misclassified images in Top-20:

| Error Categories | Numbers | Percentages(%) |
|---|---|---|
| Label May Wrong | 221 | 15.16 |
| Multiple Objects (>5) | 118 | 8.09 |
| Non-Obvious Main Object | 355 | 24.35 |
| Confusing Label | 206 | 14.13 |
| Fine-grained Label | 258 | 17.70 |
| Obvious Wrong | 234 | 16.05 |
| Partial Object | 66 | 4.53 |

https://towardsdatascience.com/review-trimps-soushen-winner-in-ilsvrc-2016-image-classification-dfbc423111dd

# Top-k accuracy analyzed

Predict:
1 *pencil box*
2 *diaper*
3 *bib*
4 *purse*
5 *running shoe*

Ground Truth:
***sleeping bag***


https://towardsdatascience.com/review-trimps-soushen-winner-in-ilsvrc-2016-image-classification-dfbc423111dd

# Top-k accuracy analyzed

Predict:
1 *dock*
2 *submarine*
3 *boathouse*
4 *breakwater*
5 *lifeboat*

Ground Truth:
***paper towel***

https://towardsdatascience.com/review-trimps-soushen-winner-in-ilsvrc-2016-image-classification-dfbc423111dd

# Top-k accuracy analyzed

Predict:
1 *bolete*
2 *earthstar*
3 *gyromitra*
4 *hen of the woods*
5 *mushroom*

Ground Truth:
***stinkhorn***

https://towardsdatascience.com/review-trimps-soushen-winner-in-ilsvrc-2016-image-classification-dfbc423111dd

# Top-k accuracy analyzed

Predict:
1 *apron*
2 *plastic bag*
3 *sleeping bag*
4 *umbrella*
5 *bulletproof vest*

Ground Truth:
***poncho***

https://towardsdatascience.com/review-trimps-soushen-winner-in-ilsvrc-2016-image-classification-dfbc423111dd

# Anomaly Detection

# What is an Anomaly?

Anomalies are hard to define in general.

Attempts at a generic definition include:

- Observations that seem to be generated by a different mechanism.
- Data instances that occur rarely and whose features differ significantly from most data.
- Outliers - observations that appear to be inconsistent (far away) with the remainder of the data

  However, inliers are not covered, that is, anomalous data that lie within the normal data distribution (say 0 in measurement results that are very likely non-zero but distributed around 0).
- Patterns in data that do not conform to a well-defined notion of normal behavior(??)

This lecture presents algorithms detecting specific data instances that may be anomalous w.r.t. the used algorithm.

It is up to the context-knowing user to decide whether such data are anomalous.

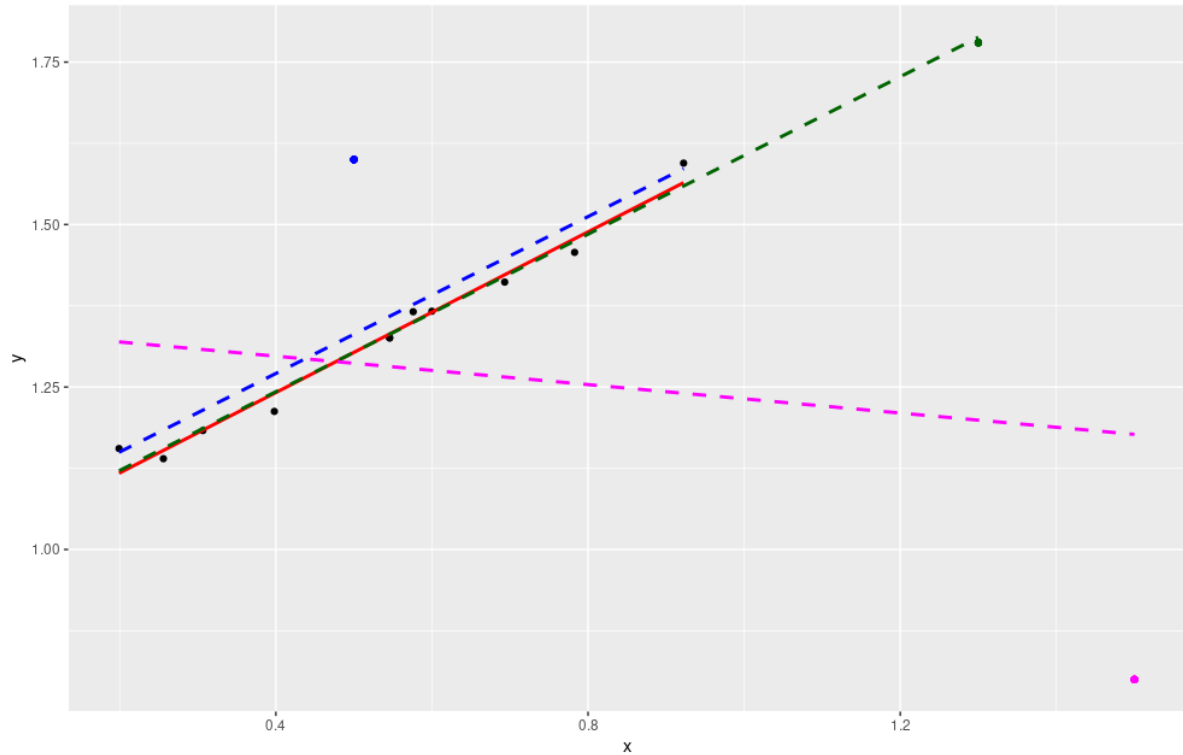
# What is the Anomaly Detection Good For?

There are two main sources of motivation for anomaly detection in machine learning:

- *Modeling perspective*: Many models are sensitive to anomalous data.
- *Application perspective*: Many tasks are based on searching for anomalies in data.

# Modeling Perspective

Recall the behavior of the linear model.




To train such a model properly, the outliers should be removed.

On the other hand, we will see that the sensitivity of some models can be used to *detect* the anomalies.

# Applications Perspective

- **Fraud Detection**
  - Analysis of purchasing behavior to detect credit card theft.
  - Identification of theft through uncharacteristic buying patterns and behavioral changes.
- **Intrusion Detection**
  - Monitoring for attacks on computer systems and networks.
  - Many attacks exhibit themselves as anomalous behavior.
- **Ecosystem Disturbances**
  - Detect atypical natural world events.
  - Prediction and understanding of hurricanes, floods, droughts, heat waves, and fires.
- **Medicine**
  - Unusual patient symptoms or test results may indicate health problems.
  - Balancing the need for further tests with the potential costs and risks.
- ...

# Causes of Anomalies in Data Preprocessing

Anomaly detection is a standard step in data preprocessing.

We typically search for anomalies from the following sources:

- Objects from different classes: Pear among apples.
- Natural variation: Abnormally tall person - not from a different class (humans) but in the sense of an extreme characteristic.
- Data measurement and collection error:
  - Thermometer positioned on the direct sun (possibly a measurement error)
  - 40 kg infant is not usual among humans (possibly a data collection error)

The anomalies mentioned above should be inspected by domain experts and possibly corrected/removed.

# Approaches to Anomaly Detection

- Model-based techniques:
  - Build a model of data.
  - Anomalies should not fit the model very well.
    For example, using a clustering model, an anomaly may lie far away from larger clusters.
  - Alternatively, removing anomalies should have the strongest impact on the model parameters (more on this later).
- Proximity-based techniques: Given distance between objects, anomalies are objects distant from others.
- Density-based techniques: Estimate the density distribution of the objects. Anomalies would probably be outside of dense regions.

I would count the most recent deep learning-based anomaly detection among the model-based techniques even though a combination of the above approaches is usually used.

We will make the above ideas more precise in the rest of this lecture.

# Anomaly Detecion Learning

There are three approaches to anomaly detection based on available information about data:

- Supervised: Given a training set distinguishing normal and anomalous data. We may train a supervised learning classifier.

  As anomalies are rare, approaches based on supervised learning are rare.
- Semi-supervised:
  - We may know what instances are normal.

    A tumor detection system trained on healthy people detects tumors as anomalies.

    In this case, we detect anomalies that are dissimilar to normal instances. We may train a model representing the normal instances and detect instances that do not fit the model.
  - We may have information about (some) normal and some anomalous instances. Here, we can use methods for semi-supervised classification.
- Unsupervised: Create a model of all instances and hope that anomalous instances will still be dissimilar to instances typical for the model.

# Issues

Task-specific issues comprise:

- Single vs. multi-attribute anomaly: The question is whether an object is anomalous due to a single attribute (200 kg person) or a combination of attributes (100 kg person of 1 meter height).

  This is especially important when data is high-dimensional. For example, in genetic data, every sample is an anomaly(?)

- Global vs. local perspective: An object may be anomalous w.r.t. all objects but not w.r.t. objects in its local neighborhood (210 cm high basketball player).
- Degree of anomaly: Usually, anomalies are reported in a binary fashion. However, we often want to measure how anomalous an object is, similar to normal objects.

Another issue is evaluation: How can we determine how good an anomaly detector is?

In the supervised case, we have a ground truth, but usually, we just observe detected anomalies.

# Various Approaches to Anomaly Detection

We shall have a look at five approaches to the unsupervised anomaly detection:

- Statistical
- Proximity-based
- Density-based
- Cluster-based
- Autoencoders

The above approaches are also used in the semi-supervised setting where we have a dataset of normal instances. However, some issues discussed further will not appear in this case.

# Statistical

## Definition 1

*An outlier is an object with a low probability in the probability distribution of the data.*

Suppose we knew the "true" probability distribution $P$ on objects. In that case, we may set up a threshold $t$ (using an appropriate training set) and say that every object $A$ with $P(A) < t$ is anomalous.

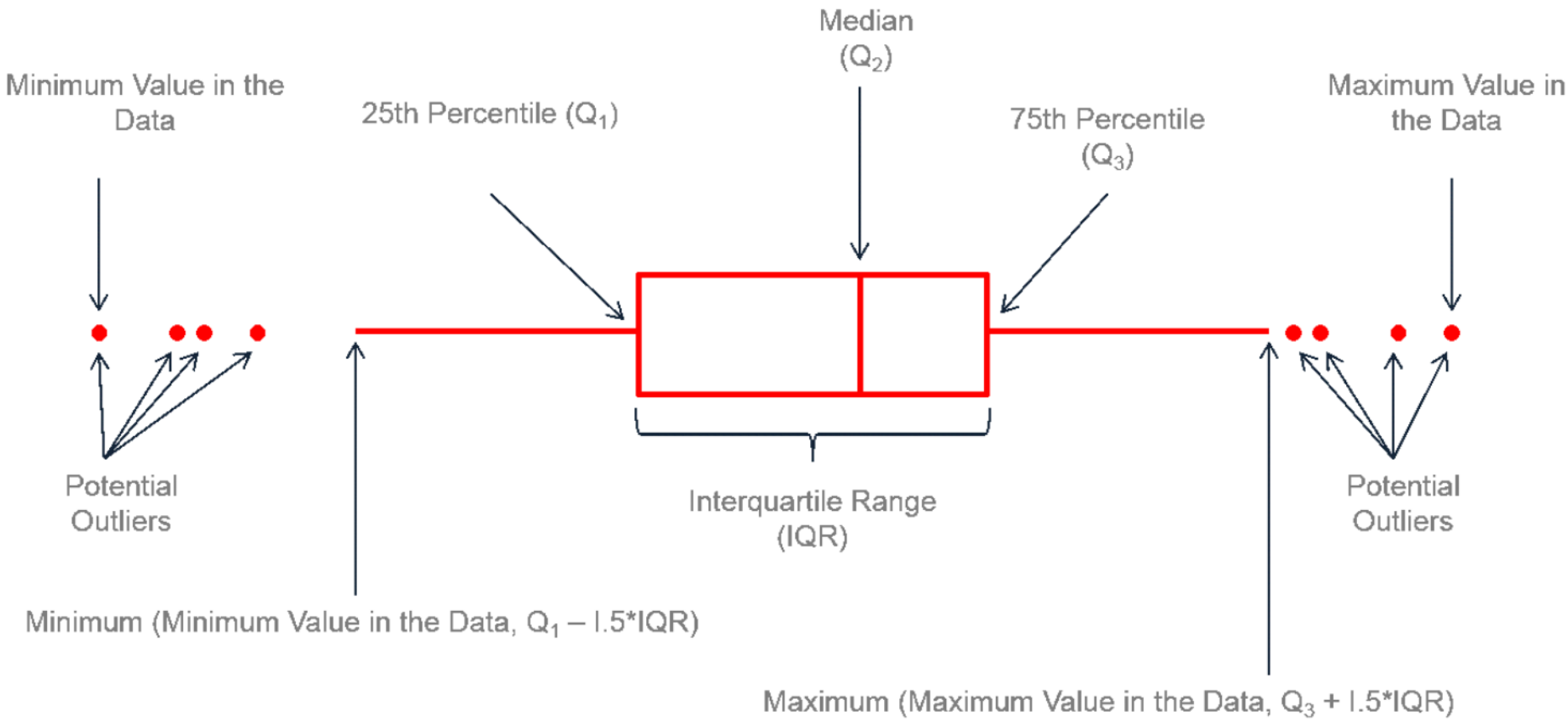
However, in most cases, we do not know $P$ and have to resort to a model of $P$ created using a dataset of feature vectors.

There are many more or less sophisticated tests based on models of $P$ in the literature.

Some use rather advanced statistical methods.

Let us have a look at a few simple examples.

# Box Plot




A simple method for outlier detection in the *univariate* case, that is, for values of a single attribute.

# Univariate Gaussian Model

Consider a numeric attribute whose values $x$ are normally distributed with mean $\mu$ and standard deviation $\sigma$.

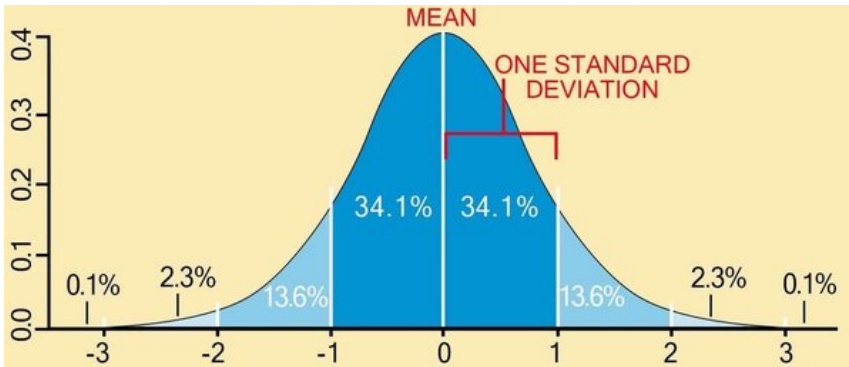
We write $N(\mu, \sigma^2)$.

Given an attribute value $x$, we compute the $z$-score:

$$z = (x - \mu)/\sigma$$

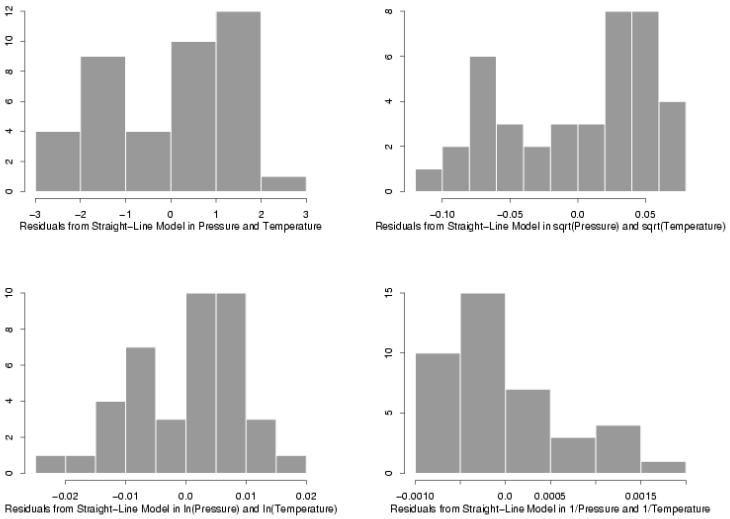
Then $z$ has the distribution $N(0, 1)$.

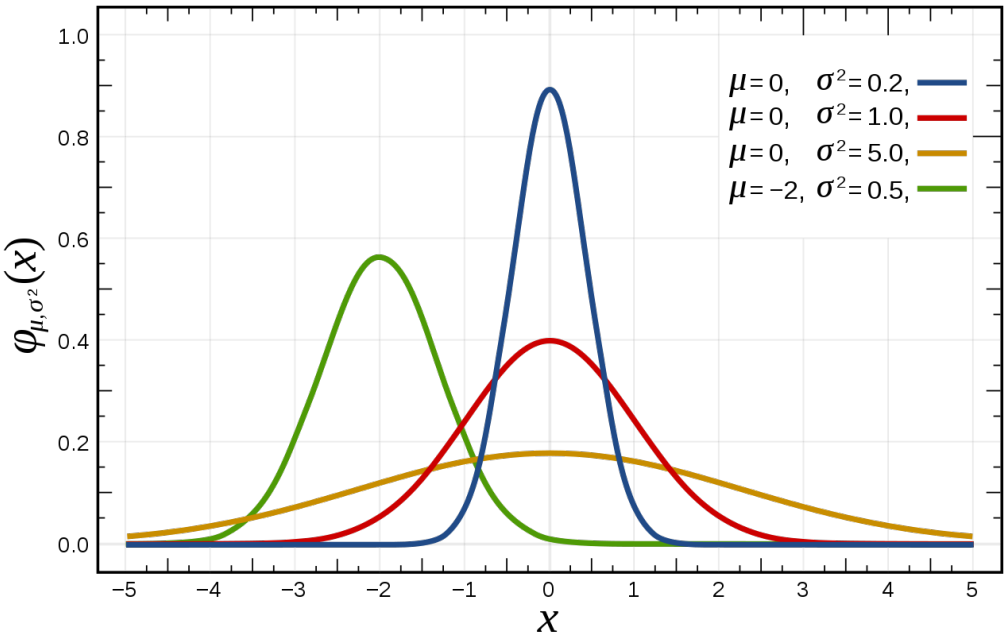
Now, choose $c$ so the probability $P(|z| \geq c)$ is small enough for $z$ to be an outlier w.r.t. $N(0, 1)$.




Given an attribute value $x$, we may decide whether $x$ is an outlier by deciding whether $|z| = |(x - \mu)/\sigma| \geq c$.

# Univariate Gaussian




**Problem 1:** The attribute does not have a normal distribution.

Before starting, use a normality test (observe the histogram, use a specialized test such as Shapiro-Wilk).

Some transformations may sometimes succeed in normalizing an attribute (Box-Cox transformation, etc.)

Different distributions can be used to model the attribute (Log Normal, Weibull, etc.), but be aware of the assumptions of anomaly detection tests!

# Univariate Gaussian

**Problem 2:** We typically do not know the population mean $\mu$ and the population variance $\sigma^2$.

They can be estimated from a dataset by the sample mean and the sample variance:

$$\bar{\mu} = \frac{1}{p}\sum_{i=1}^{p} x_i \qquad s^2 = \frac{1}{p-1}\sum_{i=1}^{p}(x_i - \bar{\mu})^2$$

However, then $z = (x - \bar{\mu})/s$ does no longer have the distribution $N(0, 1)$.

The distribution of $z$ is normal if $x$ is sampled independently of the data yielding $\bar{\mu}$ and $s^2$.

Note that $\bar{\mu}$ and $s^2$ are unbiased estimates of $\mu$ and $\sigma^2$. Also, $\bar{\mu}$ converges to $\mu$ and $s^2$ converges to $\sigma^2$ with growing sample size.

# Univariate Gaussian Model

**Problem 3:** The outliers distort the estimates $\bar{\mu}$ and $s^2$ of the mean and the standard deviation.

For example, a millionaire would not look like an outlier in a group containing a billionaire.

There is a circularity here: To get outliers using the normal distribution model, we need to remove the outliers to have a good model.

One possible heuristic is to remove the outliers one by one, starting from the most extreme, hoping the most extreme outlier will be detected in every step.

One such heuristic is the Grubb's test.

# Grubb's Test

Consider a dataset $D = \{x_1, \ldots, x_p\}$ of values of a normally distributed attribute and choose $\alpha > 0$.

The *Grubb's statistic* is defined by

$$G = \frac{\max_{i=1,\ldots,p} |x_i - \bar{x}|}{s}$$

Here $\bar{x}$ is the sample mean and $s$ sample standard deviation of $D$.

Now $P(G \geq c_{\alpha,p}) = \alpha$ if

$$c_{\alpha,p} = \frac{p-1}{\sqrt{p}} \sqrt{\frac{t^2}{p-2+t^2}}$$

Here $t$ is such a value that makes $P(T \geq t) = \alpha/(2p)$ for $T$ with the $t$-distribution with $p-2$ degrees of freedom.

For finding $t$ we used to use tables.

Grubb's test simply iteratively tests whether $G \geq c_{\alpha,p}$ and, if yes, removes an $x_i$ maximizing $|x_i - \bar{x}|$ from $D$.

# Multivariate Gaussian Model

Univariate tests cannot find all anomalies if the anomaly depends on combinations of particular attribute values.

The univariate Gaussian approach can be generalized to the multivariate Gaussian by considering the multivariate Gaussian distribution.

Mahalanobis distance:

$$\text{mahalanobis}(\vec{x}, \vec{z}) = (\vec{x} - \vec{z})\Sigma^{-1}(\vec{x} - \vec{z})^{\top}$$

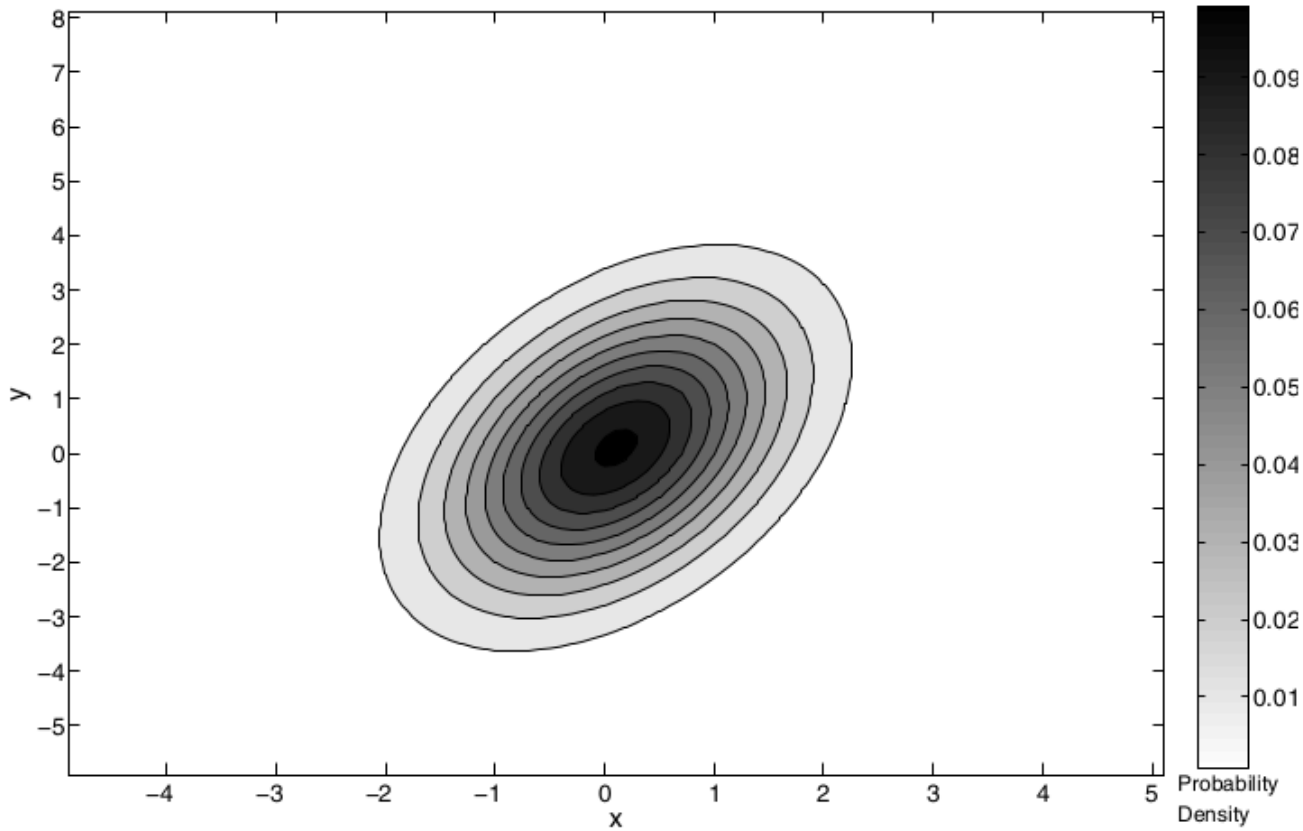
Here $\Sigma$ is the covariance matrix of the data.

Intuitively, the Mahalanobis distance generalizes the squared Euclidean distance:

$$(\vec{x} - \vec{z})(\vec{x} - \vec{z})^{\top}$$

By taking into account the "spread" of the data.

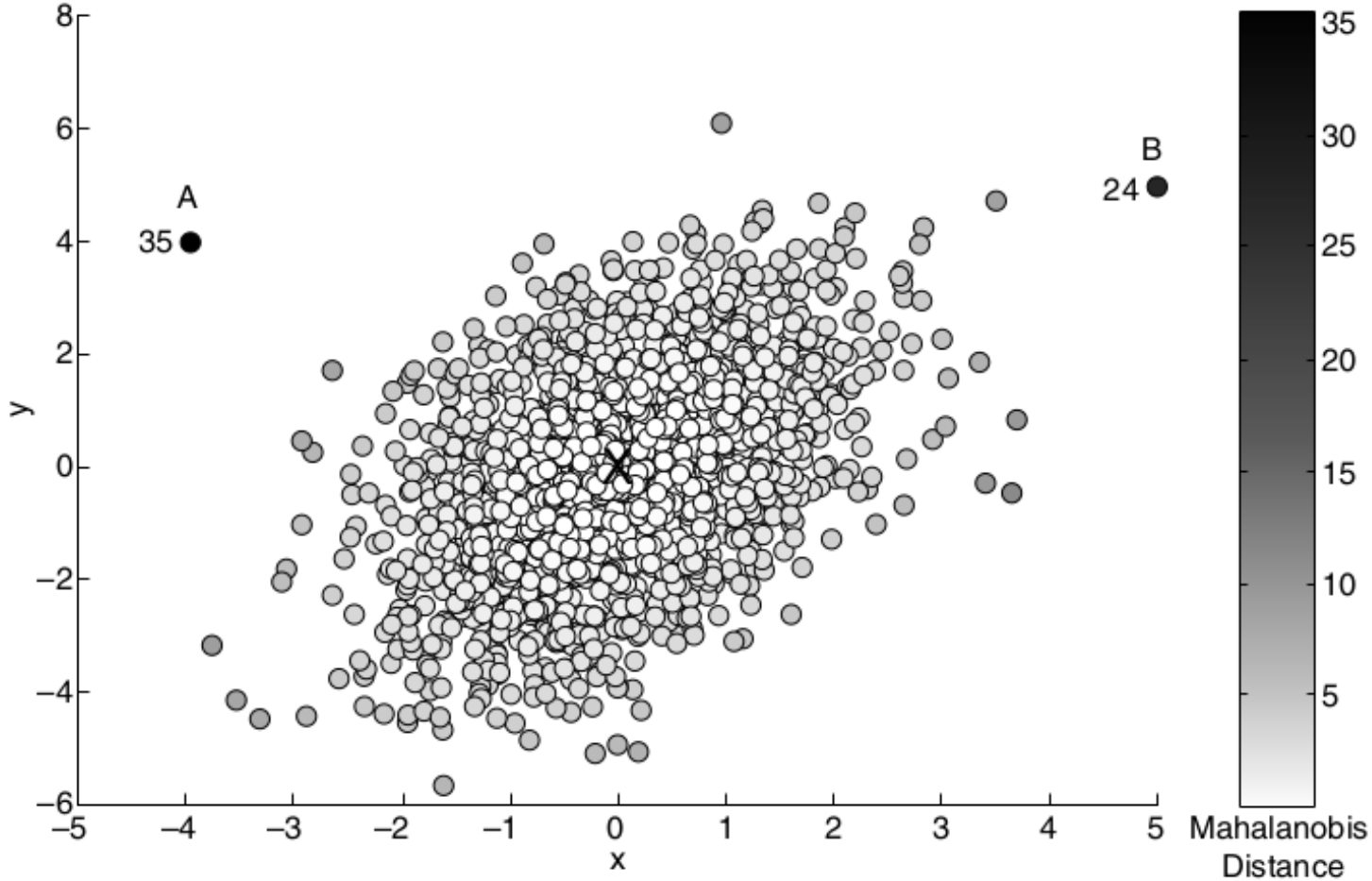
# Mahalanobis Distance




Here, we assume multivariate normal data distribution. The covariance matrix is

$$\mathbf{\Sigma} = \begin{pmatrix} 1.00 & 0.75 \\ 0.75 & 3.00 \end{pmatrix}$$

# Mahalanobis Distance




Notice that $A$ has a larger Mahalanobis distance from the cluster's center than $B$ even though it is closer in the Euclidean distance.

The reason is that the density function falls more rapidly in the direction of $A$ than in the direction of $B$.

# Likelihood-Based Detection

Assume that the dataset $D$ contains objects from a mixture of two probability distributions:

- $M$, the distribution of the majority of normal objects,
- $A$, the distribution of anomalous objects.

Let us choose $\alpha > 0$, indicating how rare the anomalies should be.

Let us assume that we have $M$ and $A$ models to compute $P_M(x)$ and $P_A(x)$ the likelihoods of generating $x$.

$A$ is often considered uniform, and $M$ is learned from the data (yes, distorted by the outliers).

Now, we compute the total likelihood of the dataset before and after removing each element. If the likelihood changes significantly, we have probably eliminated an anomaly.

Given a partition $U, V$ of $D$ we define

$$L(U,V) = \left((1-\alpha)^{|U|} \prod_{x_i \in U} P_M(x_i)\right)\left(\alpha^{|V|} \prod_{x_i \in V} P_A(x_i)\right)$$

This is the likelihood of generating $D$ using the following random process: Generate $x_1, x_2, \ldots, x_p$ sequentially and independently where each $x_i$ is generated as follows:

- Randomly choose between normal and anomaly, where the probability of choosing anomaly is $\alpha$.
- If normal has been chosen, choose $x_i$ from the distribution $M$.
- If anomaly has been chosen, choose $x_i$ from the distribution $A$.

To eliminate the "large" product, we consider the log-likelihood:

$$\begin{aligned} LL(U,V) &= \log(L(U,V)) \\ &= |U| \log(1-\alpha) + \sum_{x_i \in U} \log P_M(x_i) \\ &\quad + |V| \log \alpha + \sum_{x_i \in V} \log P_A(x_i) \end{aligned}$$

# Likelihood-Based Anomaly Detection

Start with sets $M_0 = D$ and $A_0 = \emptyset$.

The algorithm computes $M_1, M_2, \ldots$ and $A_1, A_2, \ldots$ by moving elements from $M_k$ to $A_k$ as follows:

- For every $i = 1, \ldots, p$ such that $x_i \in M_k$ compute

$$\Delta_i = |LL(M_k, A_k) - LL(M_k \setminus \{x_i\}, A_k \cup \{x_i\})|$$

- Consider $i$ maximizing $\Delta_i$. If $\Delta_i \geq c$, then

$$M_{k+1} = M_k \setminus \{x_i\} \qquad A_{k+1} = A_k \cup \{x_i\}$$

  Else, stop.

  Here, $c$ is a threshold we must set up.

Ultimately, the $A_k$ will contain the anomalies the algorithm detects.

Note that we may also move all anomalies detected in a single iteration from $M_k$ to $A_k$. This would result in a different method (also valid).

# Summary of Statistical Anomaly Detection

- Build on strong statistical foundations.
- Tests are very effective if the dataset is sufficiently large (informative).
- Lots of tests for univariate data, the area has developed for a long time.
- Fewer options for multivariate data and problematic for highly dimensional data (curse of dimensionality).

The likelihood-based approach does not assume a particular distribution shape: It can be used with arbitrary models of $P_M$ and $P_A$, including deep learning ones.

# Proximity-Based Methods

# Proximity-Based Outlier Detection

Assume a distance measure $d$. That is, given two feature vectors $\vec{x}, \vec{z}$ their distance is $d(\vec{x}, \vec{z})$.

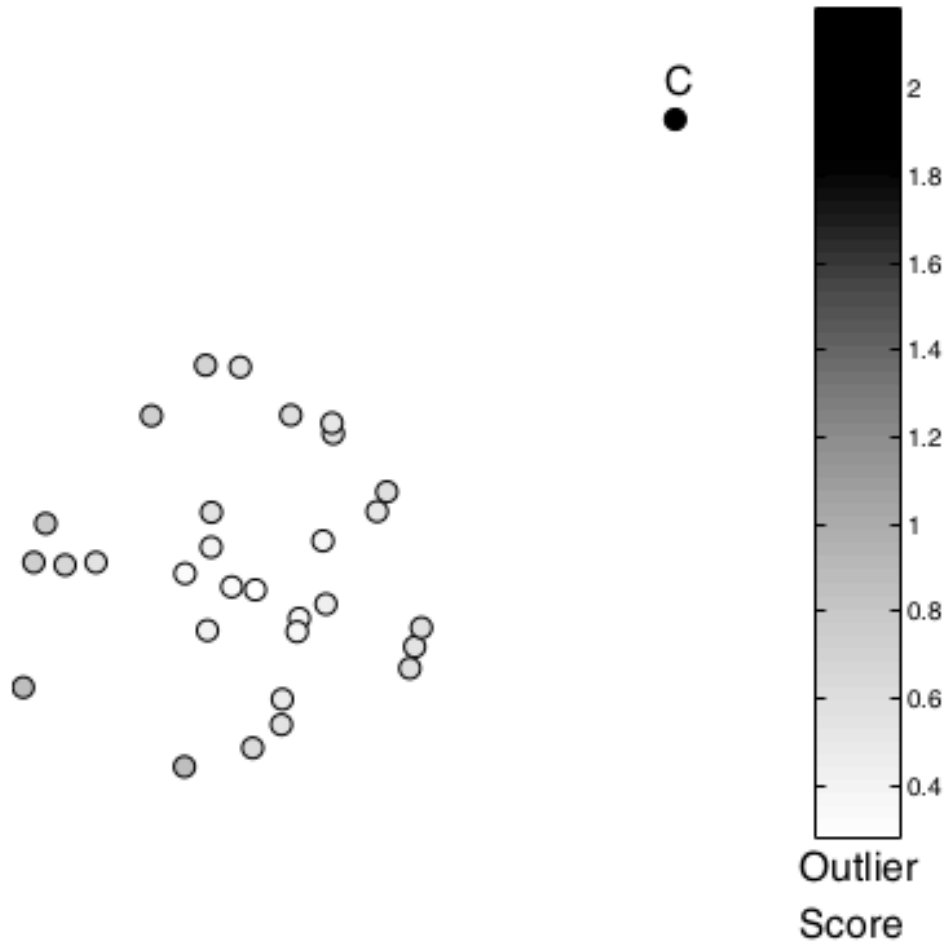
We consider the Euclidean distance for simplicity.

## Definition 2

*The outlier score of an object is given by the distance to its $k$-nearest neighbor.*

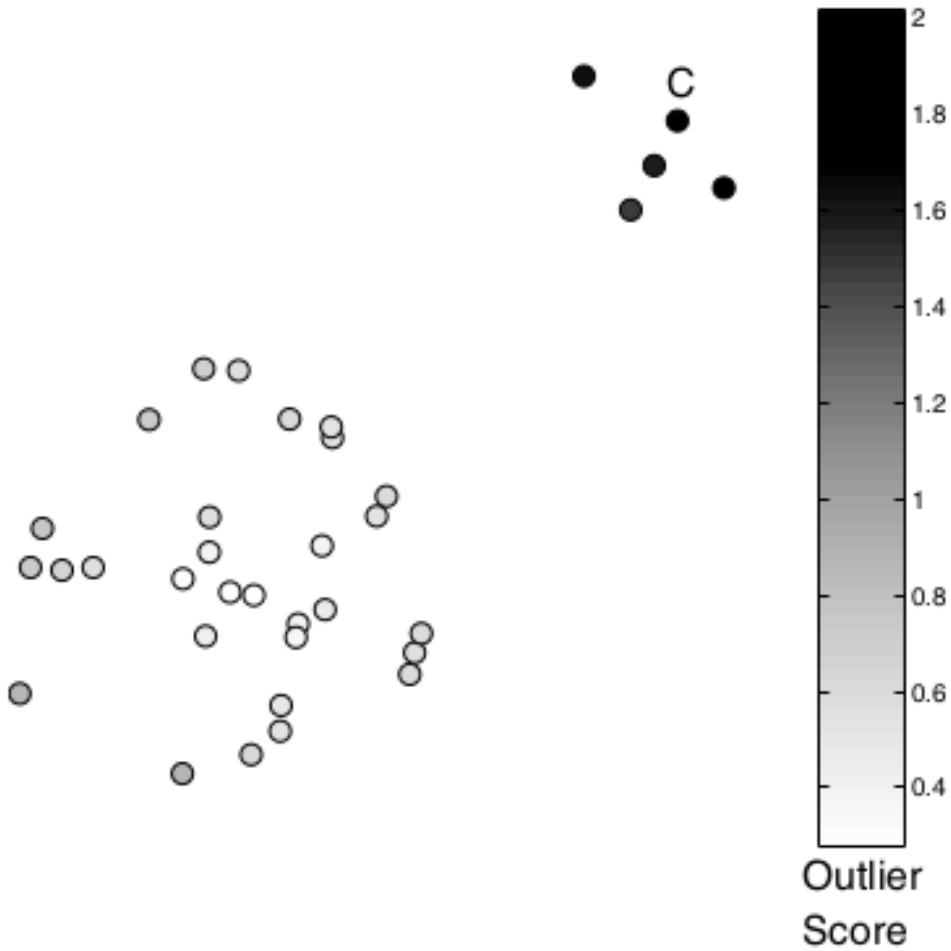
A threshold on the minimum distance of an outlier can be set on a training set.

# Outlier Score



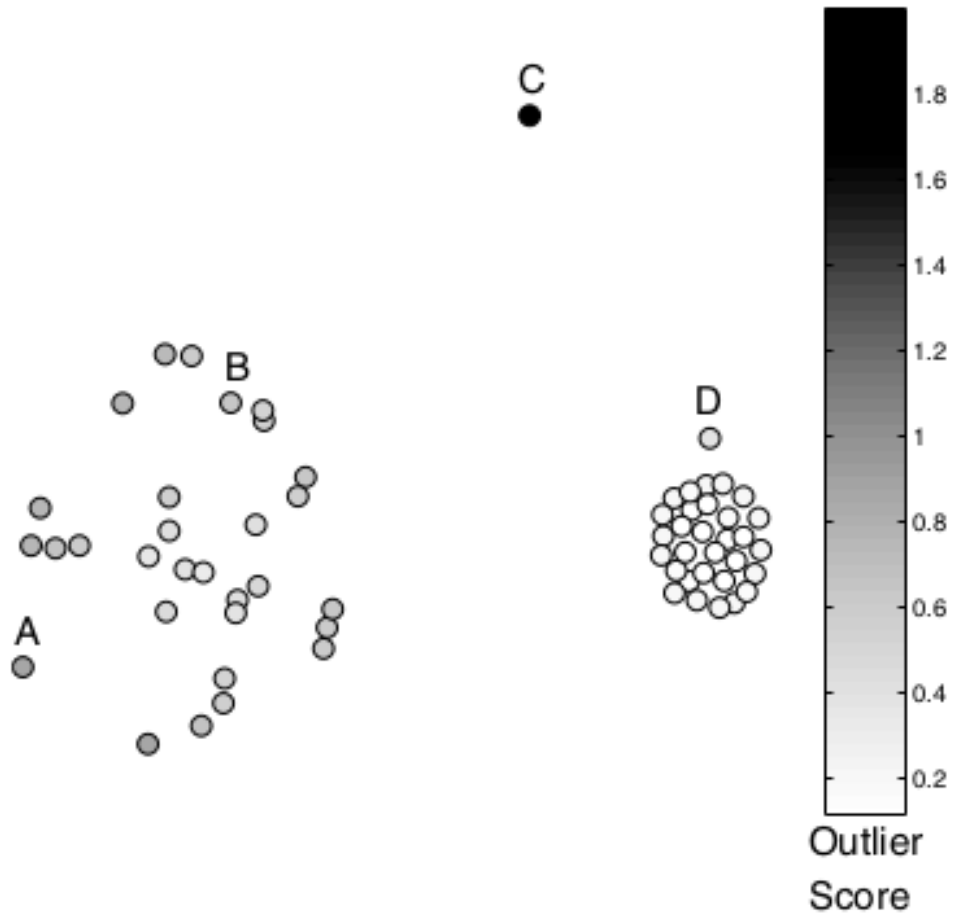

The outlier score is based on the distance to the fifth nearest neighbor.

# Outlier Score



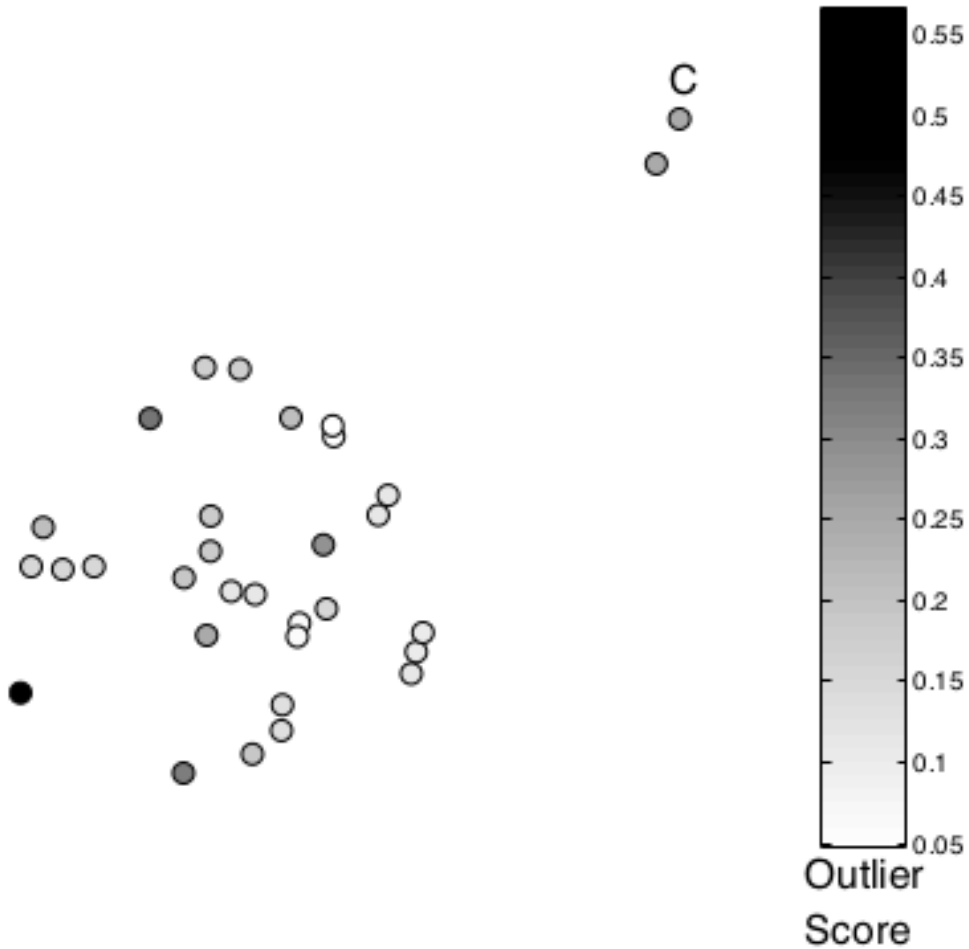

The outlier score is based on the distance to the fifth nearest neighbor.

# Outlier Score




The outlier score is based on the distance to the fifth nearest neighbor.

# Outlier Score




The outlier score is based on the distance to the first nearest neighbor.

# Proximity-Based Approaches

- Conceptually simple and easy to implement.
- Time complexity typically $\mathcal{O}(p^2)$ where $p$ is the number of feature vectors in the dataset.
- Sensitivity to the choice of parameters (the number of neighbors).
- Density/compactness not taken explicitly into account.

# Density-Based Methods

# Density-Based Approaches

## Definition 3

*The outlier score of an object is the inverse of the density around the object.*

We consider the *Local Outlier Factor (LOF)* method:

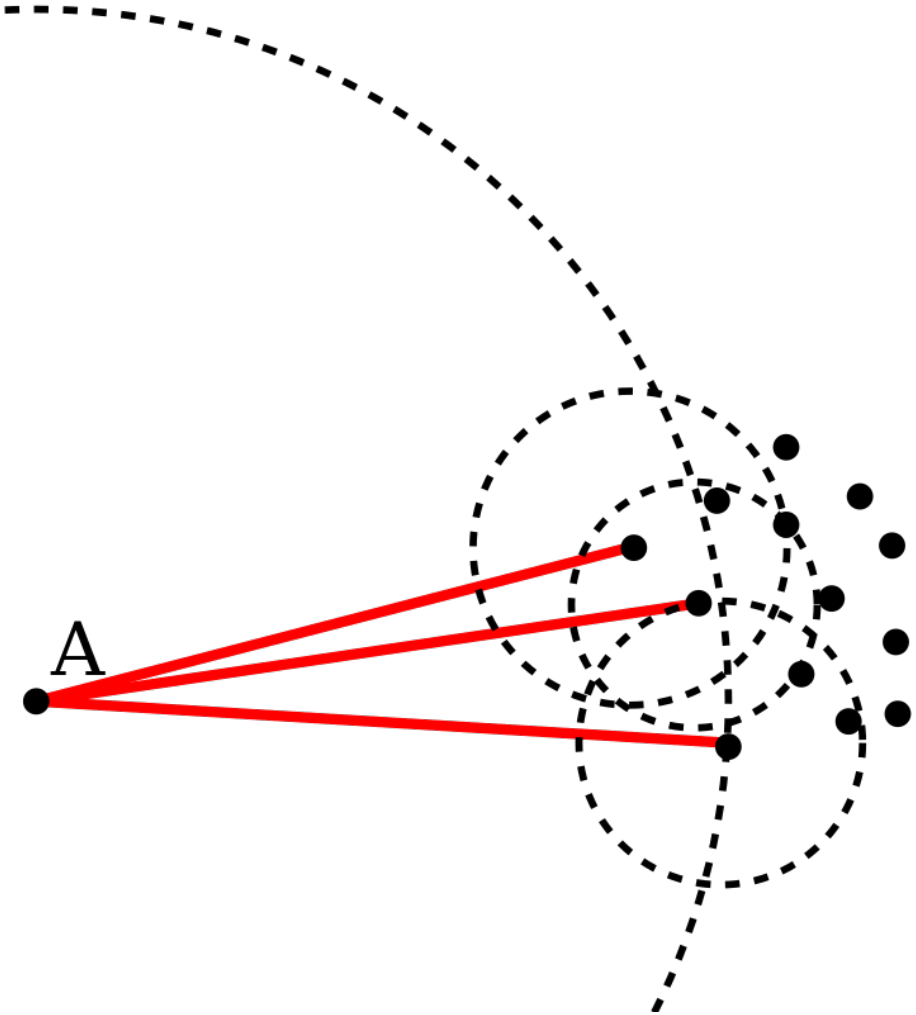
A *local density* of an object corresponds to the inverse of the average distance to its $k$-nearest neighbors.

Compute the LOF score for an object $A$ by

- computing the local density $D_A$ of $A$,
- computing the average $\bar{D}$ of the local densities of $k$-nearest neighbors of $A$,
- dividing the average local density with the local density of $A$, that is, compute $\bar{D}/D_A$.

The question is, how exactly can the local density be defined?

# LOF - Illustration




Here, the density around $A$ is smaller than the densities around the other points.

# LOF - Formally

Let us fix $k \in \mathbb{N}$.

Define $k$-distance($A$) as the distance of the $k$-th nearest neighbor.

Denote by $N_k(A)$ the set of all objects in the distance from $A$ up to $k$-distance($A$).

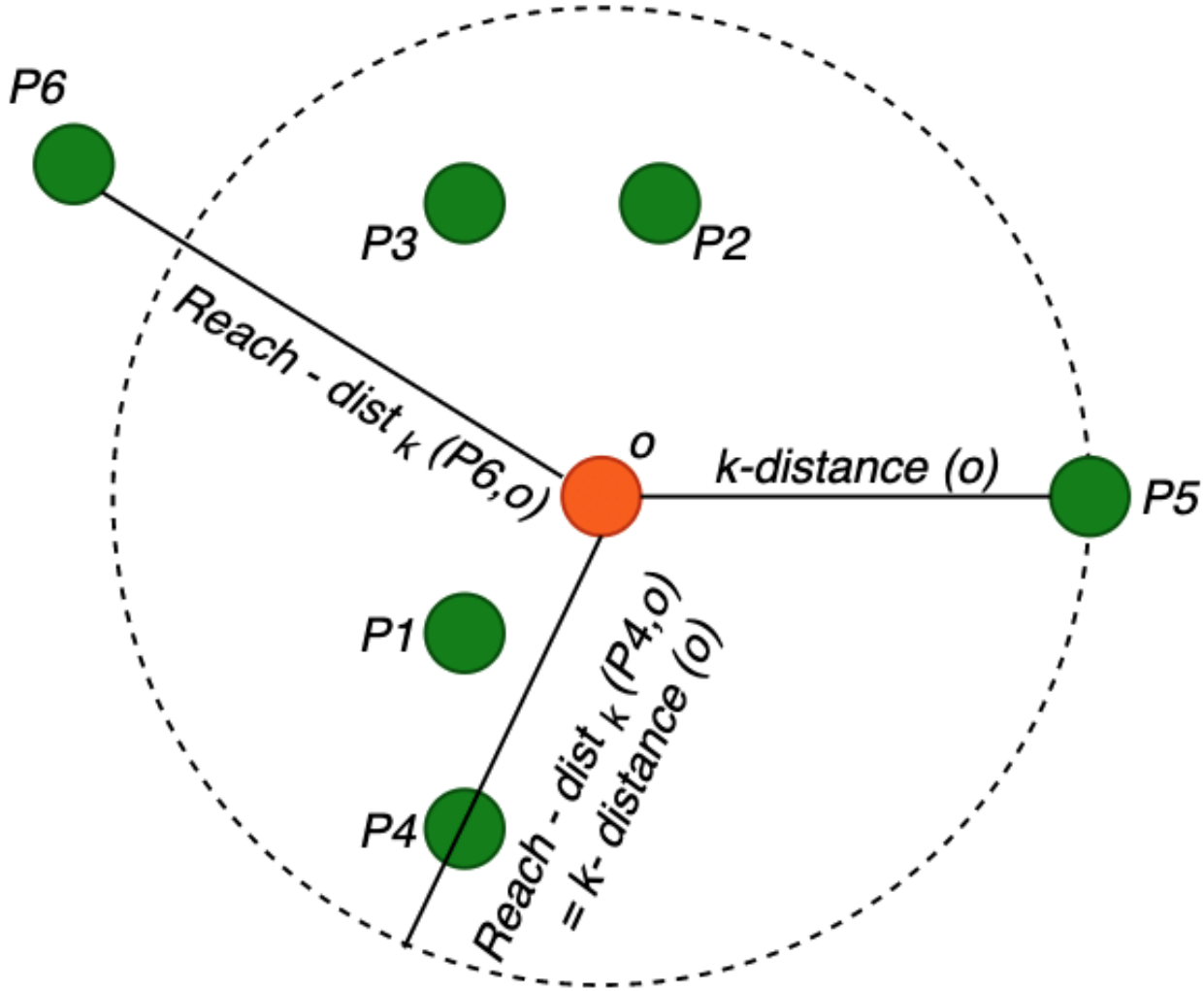
Note that if there are objects in the same distance from $A$, we may have more than $k$ elements in $N_k(A)$.

Define

$$\text{reachability-distance}_k(A, B) = \max\{k\text{-distance}(B), d(A, B)\}$$

The reachability distance of $A$ from $B$ is the distance between $A$ and $B$ but at least the distance from $B$ to the $k$-nearest neighbor.

# Reachability Distance



# LOF

Define *local reachability density*

$$\mathrm{lrd}_k(A) = \left( \frac{\sum_{B \in N_k(A)} \text{reachability-distance}_k(A, B)}{|N_k(A)|} \right)^{-1}$$

That is the reciprocal of the average reachability distance of $A$ from its $k$-nearest neighbors.

We define *local outlier factor* of an object $A$ by

$$\mathrm{LOF}_k(A) = \left( \frac{\sum_{B \in N_k(A)} \mathrm{lrd}_k(B)}{|N_k(A)|} \right) / \mathrm{lrd}_k(A)$$

Now

- $\mathrm{LOF}_k(A) \approx 1$ - similar density as the neighbors have
- $\mathrm{LOF}_k(A) < 1$ - higher density of neighbors than the neighbors have (inlier?)
- $\mathrm{LOF}_k(A) > 1$ - smaller density of neighbors than the neighbors have (outlier?)

# Example

Consider a dataset

$$D = \{A, B, C, D\} = \{5, 2, 1, 0\}$$

Consider $k = 2$ and the distance $d(U, V) = |U - V|$.

# Example

Consider a dataset

$$D = \{A, B, C, D\} = \{5, 2, 1, 0\}$$

Consider $k = 2$ and the distance $d(U, V) = |U - V|$.

---

$$N_k(A) = \{B, C\}$$

| | $k$-distance(.) | $d(A, .)$ | reach-dist$_k(A, .)$ |
|---|---|---|---|
| B | 2 | 3 | 3 |
| C | 1 | 4 | 4 |

$$\text{lrd}_k(A) = \left(\frac{1}{2}\left(\text{reach-dist}_k(A, B) + \text{reach-dist}_k(A, C)\right)\right)^{-1} = 2/7$$

# Example

Consider a dataset

$$D = \{A, B, C, D\} = \{5, 2, 1, 0\}$$

Consider $k = 2$ and the distance $d(U, V) = |U - V|$.

---

$$N_k(B) = \{C, D\}$$

| | $k$-distance(.) | $d(B, .)$ | reach-dist$_k(B, .)$ |
|---|---|---|---|
| C | 1 | 1 | 1 |
| D | 2 | 2 | 2 |

$$\text{lrd}_k(B) = \left(\frac{1}{2}\left(\text{reach-dist}_k(B, C) + \text{reach-dist}_k(B, D)\right)\right)^{-1} = 2/3$$

# Example

Consider a dataset

$$D = \{A, B, C, D\} = \{5, 2, 1, 0\}$$

Consider $k = 2$ and the distance $d(U, V) = |U - V|$.

---

$$N_k(C) = \{B, D\}$$

| | $k$-distance(.) | $d(C, .)$ | reach-dist$_k(C, .)$ |
|---|---|---|---|
| B | 2 | 1 | 2 |
| D | 2 | 1 | 2 |

$$\text{lrd}_k(C) = \left(\frac{1}{2}\left(\text{reach-dist}_k(C, B) + \text{reach-dist}_k(C, D)\right)\right)^{-1} = 1/2$$

# Example

Consider a dataset

$$D = \{A, B, C, D\} = \{5, 2, 1, 0\}$$

Consider $k = 2$ and the distance $d(U, V) = |U - V|$.

---

$$N_k(D) = \{B, C\}$$

| | $k$-distance(.) | $d(D, .)$ | reach-dist$_k(D, .)$ |
|---|---|---|---|
| B | 2 | 2 | 2 |
| C | 1 | 1 | 1 |

$$\text{lrd}_k(D) = \left(\frac{1}{2}\left(\text{reach-dist}_k(D, B) + \text{reach-dist}_k(D, C)\right)\right)^{-1} = 2/3$$

# Example

$$\text{lrd}_k(A) = 2/7$$
$$\text{lrd}_k(B) = 2/3$$
$$\text{lrd}_k(C) = 1/2$$
$$\text{lrd}_k(D) = 2/3$$

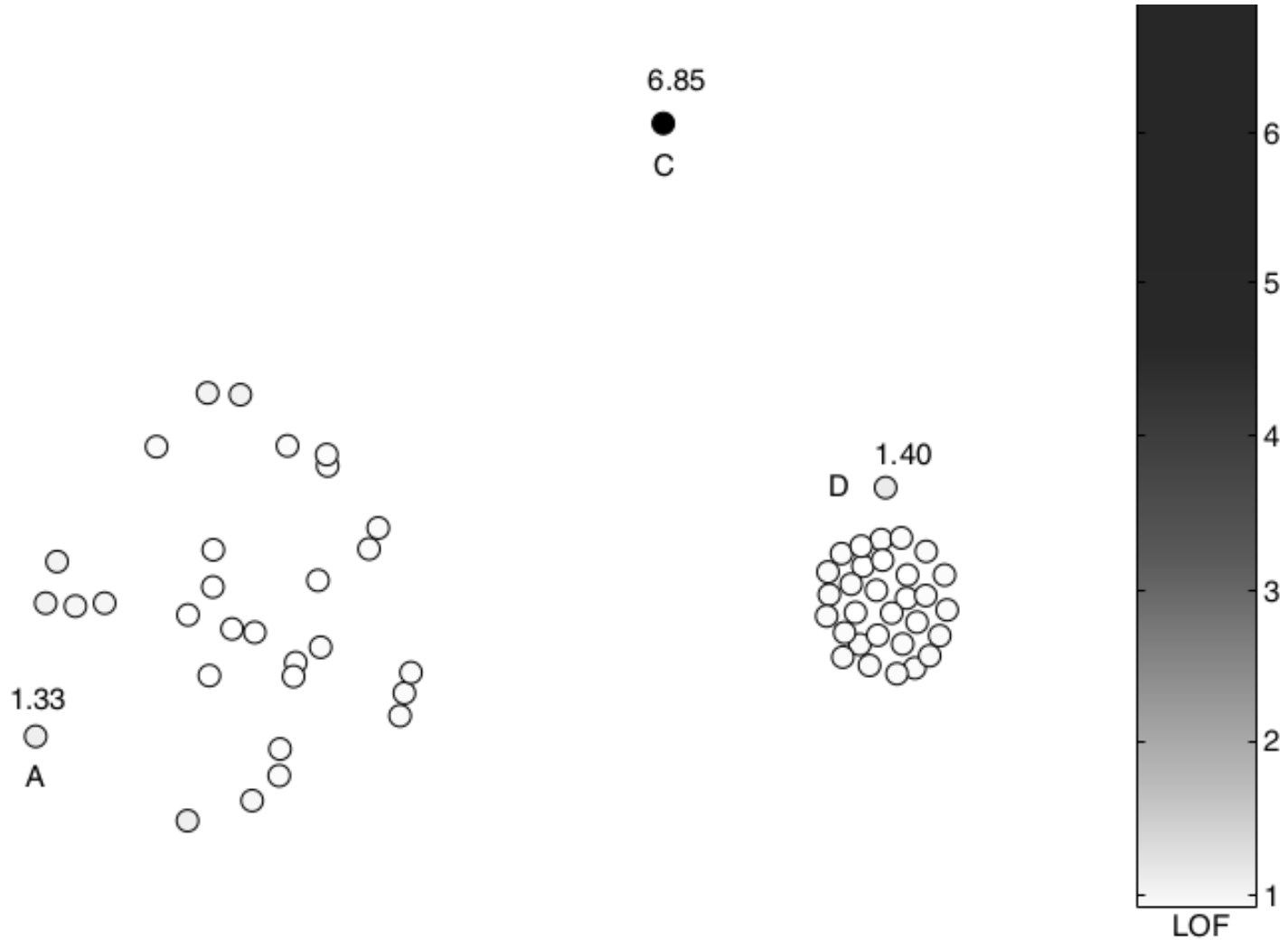
$$\begin{aligned}\text{LOF}_k(A) &= \frac{1}{2}(\text{lrd}_k(B) + \text{lrd}_k(C))/\text{lrd}_k(A) \\ &= (1/2)(2/3 + 1/2)/(2/7) = 2.041\end{aligned}$$

$$\begin{aligned}\text{LOF}_k(B) &= \frac{1}{2}(\text{lrd}_k(C) + \text{lrd}_k(D))/\text{lrd}_k(B) \\ &= (1/2)(1/2 + 2/3)/(2/3) = 0.875\end{aligned}$$

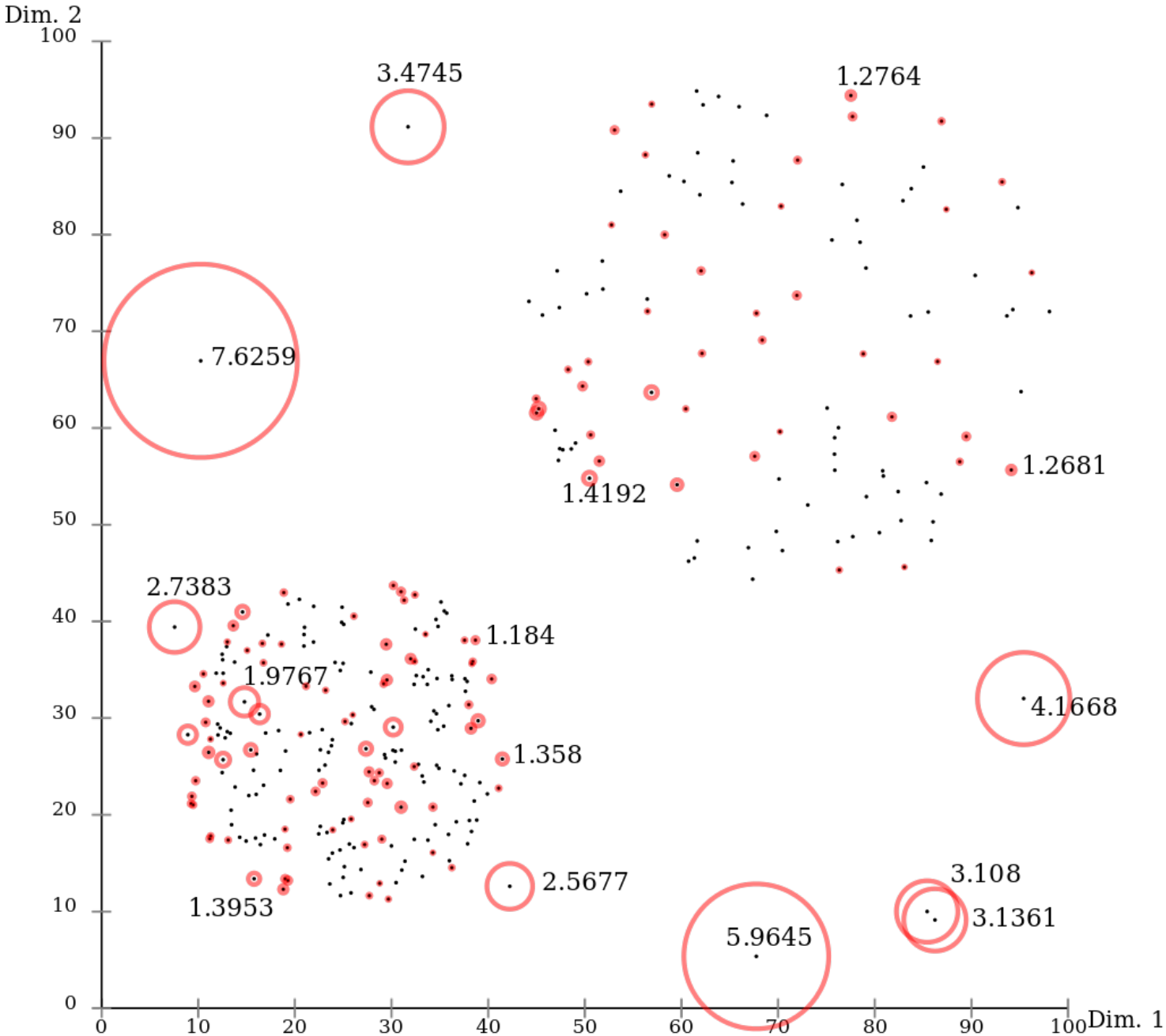
$$\begin{aligned}\text{LOF}_k(C) &= \frac{1}{2}(\text{lrd}_k(B) + \text{lrd}_k(D))/\text{lrd}_k(B) \\ &= (1/2)(2/3 + 2/3)/(2/3) = 1\end{aligned}$$

$$\text{LOF}_k(D) = 0.875$$

# Another Example



# Yet Another Example



# Comments on Density Based Methods

- As opposed to the distance-based methods, quantifies the distance of the neighborhood of the (potential) outliers.
- Time complexity still $\mathcal{O}(p^2)$ but may be reduced for low dimensional data (to $\mathcal{O}(p \log p)$) using special data structures.
- Still need to determine the parameter $k$.

# Clustering-Based Methods

# Clustering Approach

Clustering is supposed to group similar objects.

Anomaly detection detects objects dissimilar to others.

So, clustering inherently solves similar problems.

But how do we detect anomalies based on clustering?

We may detect anomalies as elements of small clusters.
This approach is problematic as it strongly depends on the size/number of clusters.

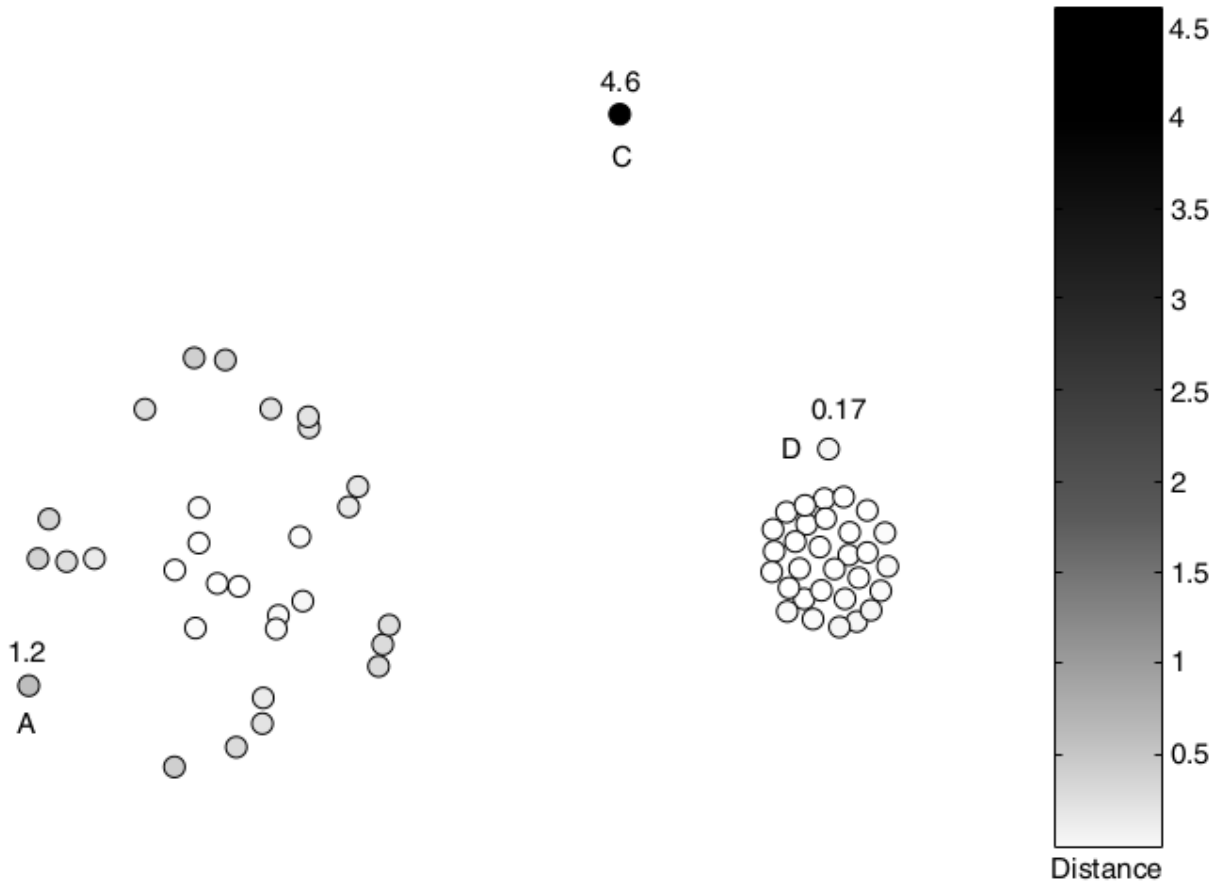
A better approach would be to asses how strongly objects belong to clusters.

## Definition 4

*An object is a cluster-based outlier if the object does not strongly belong to any cluster.*

Depending on the particular clustering algorithm, we have (at least some) information about the cohesion of objects.

# Anomaly Detection Using $k$-Means Clustering

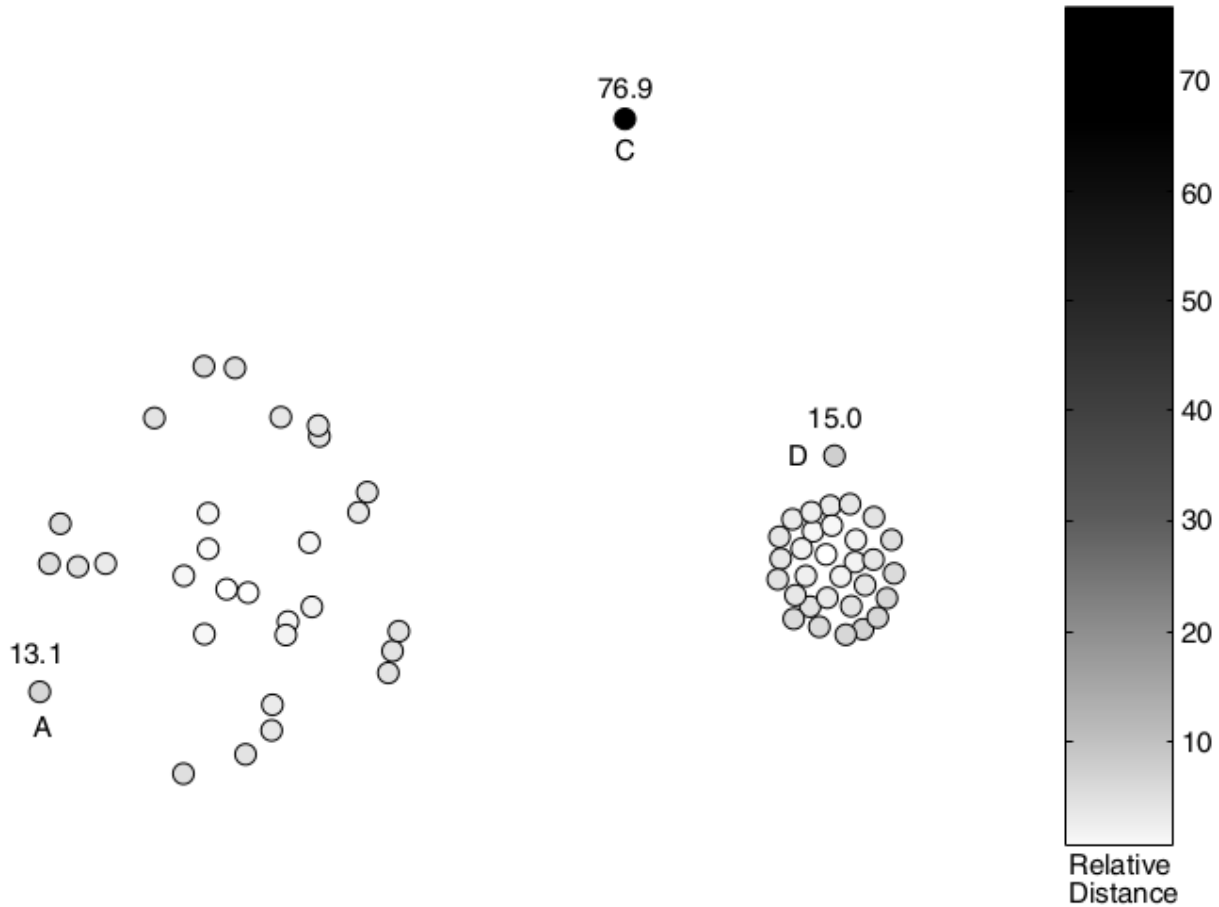



In $k$-means, the clusters are represented by centroids.

We may measure the anomaly of a given object using the distance to its cluster centroid.

This approach does not take into account the density of clusters.

# Anomaly Detection Using $k$-Means Clustering




Here, we compute the relative distance to the cluster center, that is, the ratio of the object's distance to the centroid and the median distance of all objects of the same cluster to the centroid.

# Clustering Based Anomaly Detection

Other algorithms may provide different information (probability density of the clusters, etc.).

The usual problem: Outliers have an impact on the clustering itself.

Some algorithms can be modified to treat potential outliers specially.

For example, during the $k$-means clustering, put objects very far away from the current cluster centroids into a special category not used to move the centroids. After every step, test whether some of these objects became close enough to at least one centroid (in which case these objects will be removed from the special category).

Setting the proper number of clusters is an issue as well.

For anomaly detection, a larger number of smaller clusters may be beneficial as they may be more cohesive. If an object seems to be an outlier with small clusters, it is probably an outlier.

# Comments on Clustering Based Methods

Some clustering methods have a sub-quadratic complexity.

Conceptually, clustering complements anomaly detection, so it is natural to compute both together.

On the other hand, the results strongly depend on the number of clusters, and anomalies may distort the clustering.

# Autoencoders as Anomaly Detectors

... just a short comment

# Neural Networks in Anomaly Detection

The previous approaches work well with (relatively) low-dimensional data.

However, what should we do with data with high or even variable dimensions?

- Images
- Text
- Video
- Semantic graphs
- ...

One way is to transform them into lower-dimensional data.

Train a neural network that takes the large input and returns a smaller representation, which (hopefully) preserves crucial features of the input.

# Autoencoders

An autoencoder consists of two parts:

- $\phi : \mathbb{R}^n \to \mathbb{R}^m$ the encoder
- $\psi : \mathbb{R}^m \to \mathbb{R}^n$ the decoder

The goal is to find $\phi$, $\psi$ so that $\psi \circ \phi$ is (almost) identity.

The value $\vec{h} = \phi(\vec{x})$ is called the *latent representation* of $\vec{x}$.

**Training:** Assume

$$\mathcal{T} = \{\vec{x}_1, \ldots, \vec{x}_p\}$$

where $\vec{x}_i \in \mathbb{R}^n$ for all $i \in \{1, \ldots, n\}$.

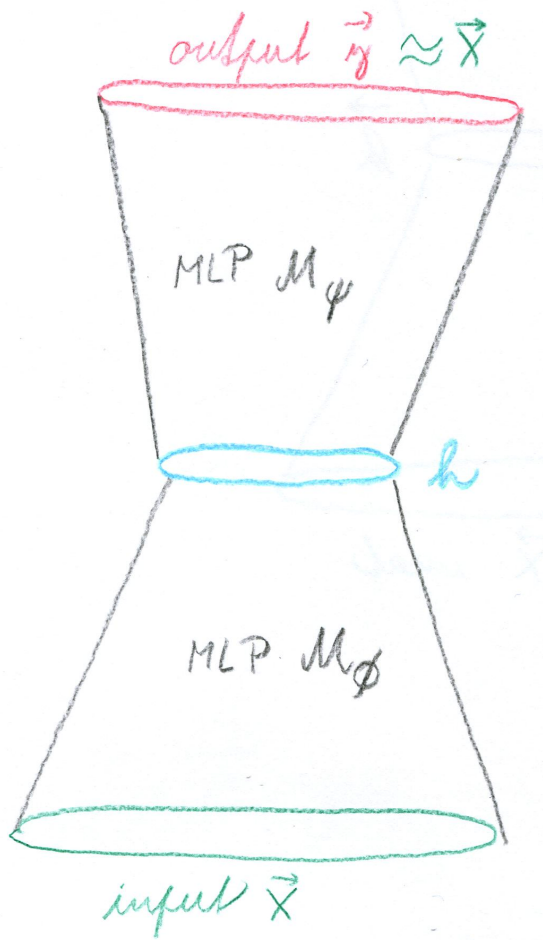
Minimize the **reconstruction** error

$$E = \sum_{i=1}^{p} (\vec{x}_i - \psi(\phi(\vec{x}_i)))^2$$

# Autoencoders – neural networks

Both $\phi$ and $\psi$ can be represented using MLP $\mathcal{M}_\phi$ and $\mathcal{M}_\psi$, respectively.

$\mathcal{M}_\phi$ and $\mathcal{M}_\psi$ can be connected into a single network.



# Autoencoders – Usage

- Compression/feature extraction – from $\vec{x}$ to $\vec{h}$.
- Dimensionality reduction – the latent representation $\vec{h}$ has a smaller dimension.
- Generative versions – (roughly) generate $\vec{h}$ from a known distribution, let $\mathcal{M}_\psi$ generate realistic inputs/outputs $\vec{x}$
- **Anomaly detection** (see the next slide)

# Anomaly Detection with Autoencoders

Straightforward approach:

- Train the autoencoder on the normal data.
- Detect an anomaly using the large reconstruction error.

  The idea is that an anomaly will not be properly reconstructed.

More general approach:

- Train the autoencoder and transform the normal data to their low dimensional latent representations.
- Use one of the previous approaches to anomaly detection to detect anomalies in the latent representations.

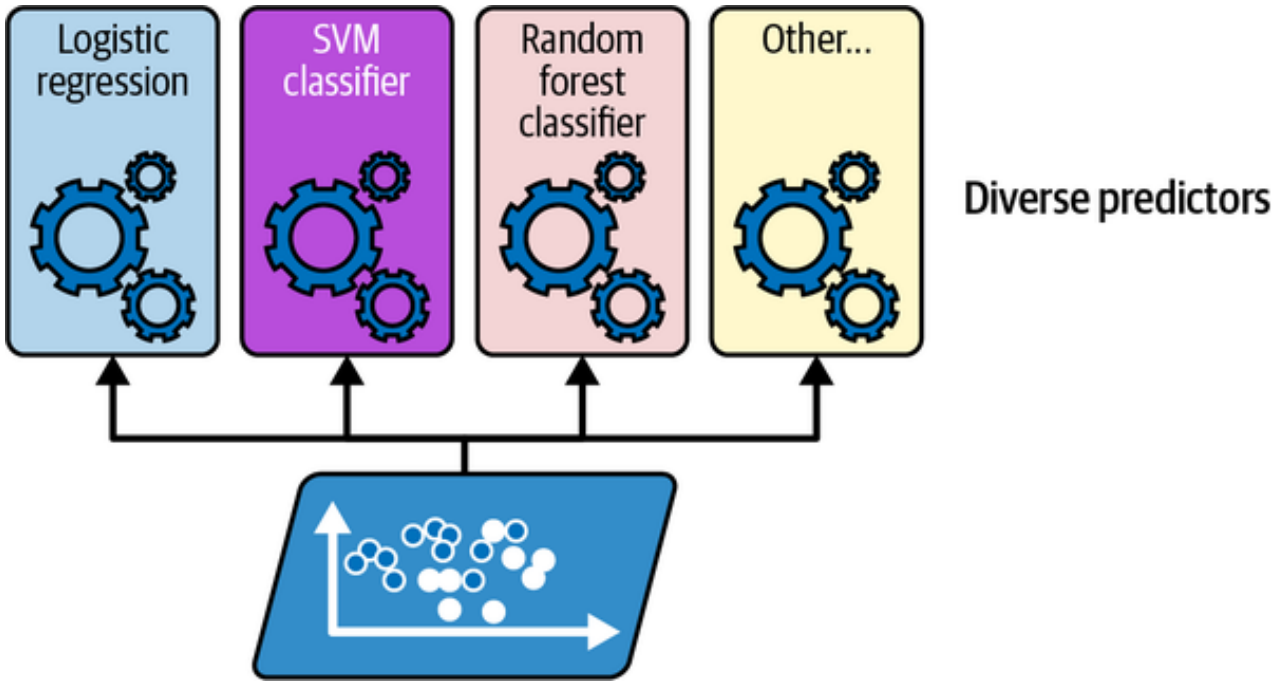
  The assumption is that the latent representation of an anomaly will substantially differ from the latent representations of normal instances.

# Summary of Anomaly Detection

- It is hard even to define an anomaly - context knowledge is usually needed.
- Supervised anomaly detection reduces to classification.
- Unsupervised anomaly detection can be done in various ways; we have seen the following:
  - Statistical
  - Proximity-based
  - Density-based
  - Cluster-based
  - Autoencoders
- Semi-supervised anomaly detection is usually concerned with data where the normal class is known.
- There are many more methods: One class SVM, isolation forests, etc.

# Ensemble Methods

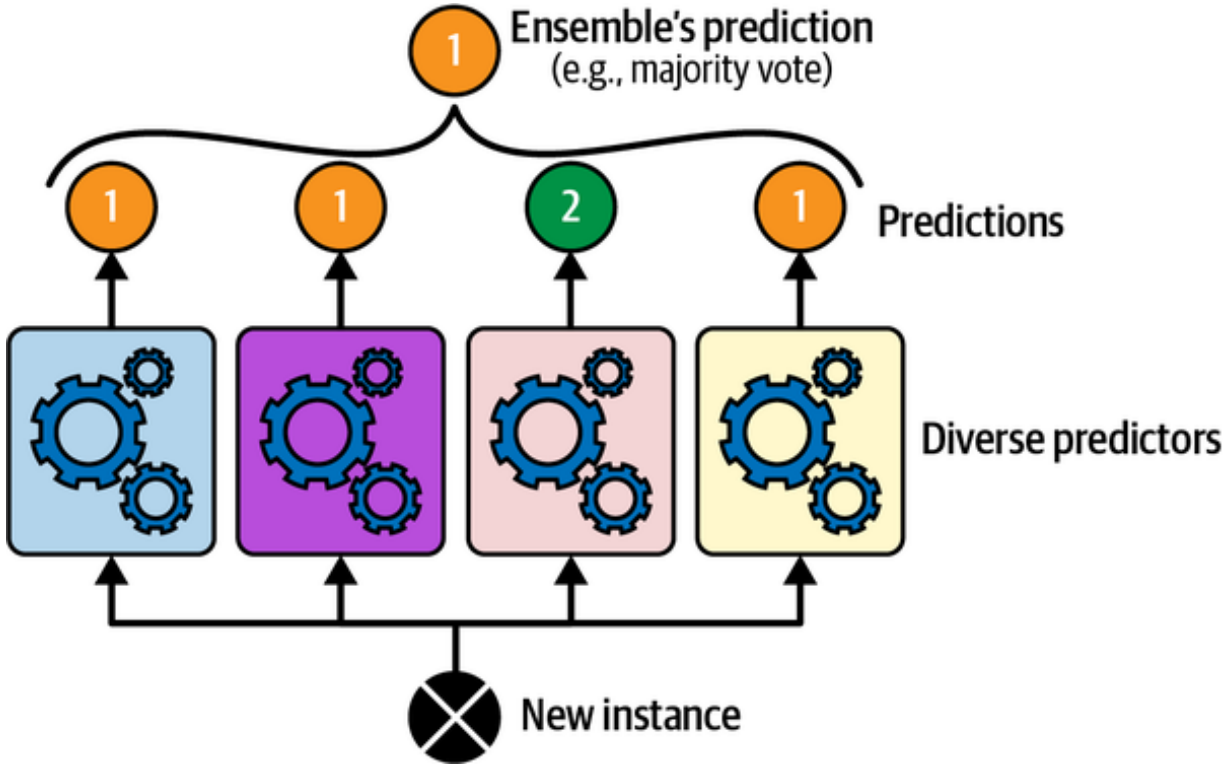
# Voting Classifiers




Train several models. They may differ in

- Structure (completely different models)
- Training data (subsample the training set)

# Voting Classifiers




During the inference, ensemble the predictions. For binary classifiers, you may do the following:

- Take the majority vote.
- Summarize the output probabilities (e.g., by averaging)
- If logistic regression or neural networks with logistic output activation are used, summarize the outputs before the application of the last logistic sigmoid.

# The Ensemble Idea

An ensemble of *weak* classifiers may be a *strong classifier*.

**Analogy:** Assume a coin toss that has a 51 percent chance of heads and 49 percent tails.

Try 1,000 tosses in a row. Do this repeatedly. In 75 percent of cases, you get more heads than tails. With 10,000 tosses, it would be 97 percent.

Intuitively, we might suspect that a voting ensemble of 1,000 sufficiently independent classifiers, each with an accuracy of around 51 percent, would have an accuracy of around 70 percent.

The ensemble methods work best when the models are as independent as possible.

Use different learning methods or independently chosen training sets (see later slides).

Let us try to formalize the above intuition.

# Bias/Variance Decomposition

Consider a random function $g(x) + \varepsilon$ where $g : \mathbb{R} \to \mathbb{R}$ and $\varepsilon$ is random noise with mean 0 and variance $\sigma^2$.

Consider a dataset for regression:

$$D = \{(x_1, d_1), \ldots, (x_p, d_p)\}$$

Here $x_1, \ldots, x_p$ are fixed inputs, and each $d_k = g(x_k) + \varepsilon_k$ where $\varepsilon_1, \ldots, \varepsilon_p$ are independent samples from the noise $\varepsilon$.

We get different models for different samples of the training set $D$.

We may ask:

- How much do the models trained on different samples differ?
- How much do they differ (on average) from $g$?

Let us denote by $h_D : \mathbb{R} \to \mathbb{R}$ a model trained on the dataset $D$ by minimizing the squared error $\sum_{k=1}^{p} (h_D(x_k) - d_k)^2$.

# Bias/Variance Decomposition

Let us consider the expected value of the squared error:

$$\mathbb{E}\left(\sum_{k=1}^{p}(h_D(x_k)-d_k)^2\right)=\sum_{k=1}^{p}\mathbb{E}\,(h_D(x_k)-d_k)^2$$

The expectation is computed over the distribution of the datasets $D$.

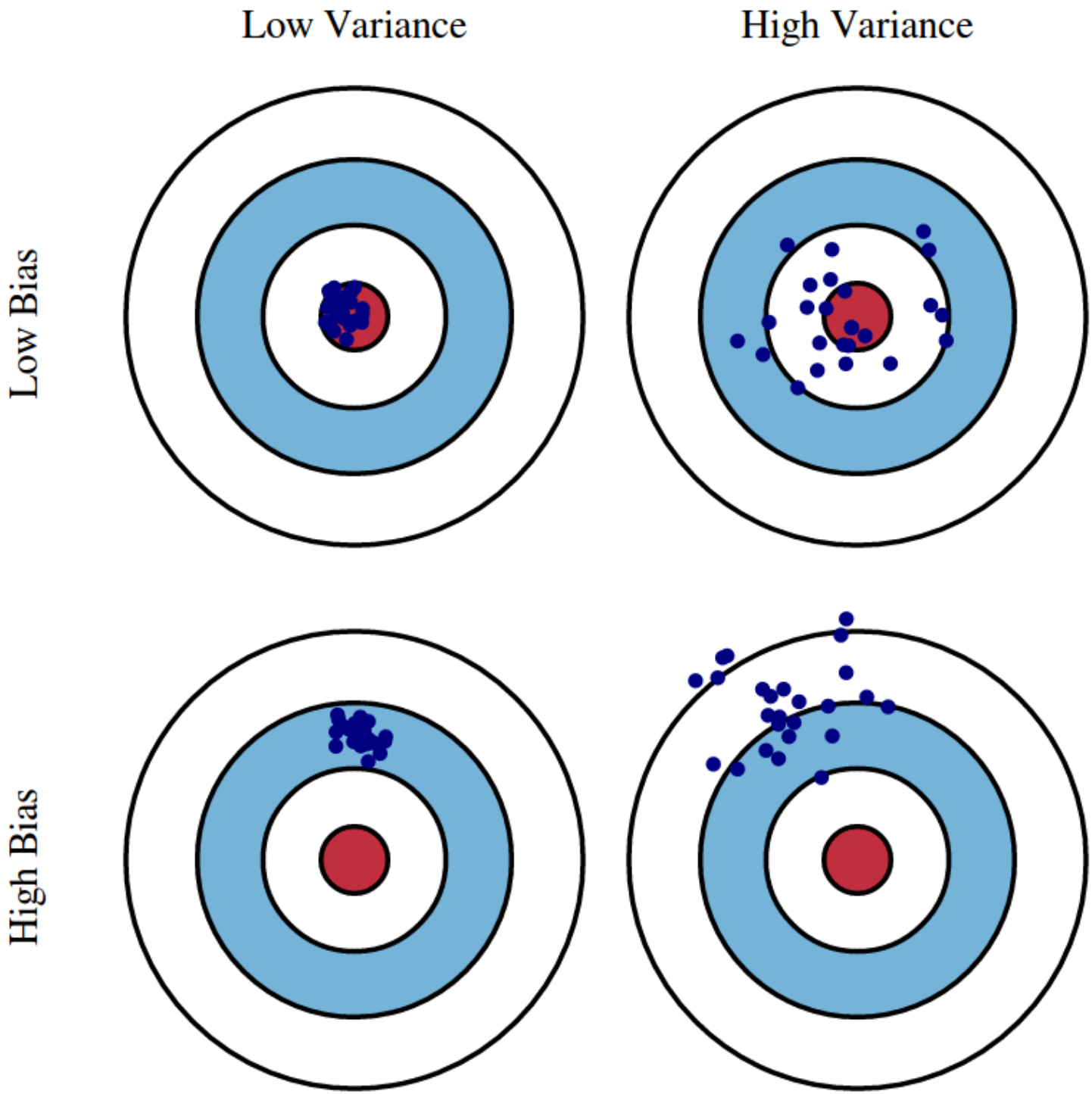
One can prove that

$$\sum_{k=1}^{p}\mathbb{E}\,(h_D(x_k)-d_k)^2=$$

$$\sum_{k=1}^{p}\mathbb{E}\,(h_D(x_k)-\mathbb{E}[h_D(x_k)])^2+(\mathbb{E}[h_D(x_k)]-g(x_k))^2+\sigma^2$$

Here

- $\mathbb{E}\,(h_D(x_k)-\mathbb{E}[h_D(x_k)])^2$ is the *Variance* of the model on $x_k$
  how much the model's value jumps around its average on $x_k$.
- $\mathbb{E}[h_D(x_k)]-g(x_k)$ is the *Bias* of the model.
  how much the average model's value differs from the true values.
- $\sigma^2$ is the noise variance.


Low Variance
High Variance
Low Bias
High Bias

# B/V Decomposition Proof (Optional)

Let us study $\mathbb{E}\left(h_D(x_k) - d_k\right)^2$. To simplify notation we drop the index $k$ and write $\mathbb{E}_D\left(h_D(x) - d\right)^2$.

$$\begin{aligned}
\left(h_D(x) - d\right)^2 &= \left(h_D(x) - \mathbb{E}[h_D(x)] + \mathbb{E}[h_D(x)] - d\right)^2 \\
&= \left(h_D(x) - \mathbb{E}[h_D(x)]\right)^2 \\
&\quad + \left(\mathbb{E}[h_D(x)] - d\right)^2 \\
&\quad + 2\left(h_D(x) - \mathbb{E}[h_D(x)]\right)\left(\mathbb{E}[h_D(x)] - d\right)
\end{aligned}$$

Now, just apply $\mathbb{E}$ to both sides. As

$$\mathbb{E}(d) = \mathbb{E}(g(x) + \varepsilon) = g(x)$$

$$\mathbb{E}(d^2) = \mathbb{E}(g(x) + \varepsilon)^2 = \mathbb{E}(g(x)^2 + 2\varepsilon g(x) + \varepsilon^2) = g(x)^2 + \sigma^2$$

We obtain ...

# B/V Decomposition Proof (Optional)

... this

$$
\begin{aligned}
&\mathbb{E}\left(\mathbb{E}[h_D(x)] - d\right)^2 \\
&\quad = \mathbb{E}\left(\mathbb{E}[h_D(x)]^2 - 2d\mathbb{E}[h_D(x)] + d^2\right) \\
&\quad = \mathbb{E}[h_D(x)]^2 - 2\mathbb{E}(d)\mathbb{E}[h_D(x)] + \mathbb{E}d^2 \\
&\quad = \mathbb{E}[h_D(x)]^2 - 2g(x)\mathbb{E}[h_D(x)] + g(x)^2 + \sigma^2 \\
&\quad = \mathbb{E}\left(\mathbb{E}[h_D(x)] - g(x)\right)^2 + \sigma^2
\end{aligned}
$$

and this

$$
\begin{aligned}
&\mathbb{E}\left[2\left(h_D(x) - \mathbb{E}[h_D(x)]\right)\left(\mathbb{E}[h_D(x)] - d\right)\right] \\
&\quad = \mathbb{E}\left[2\left(h_D(x) - \mathbb{E}[h_D(x)]\right)\left(\mathbb{E}[h_D(x)] - g(x) - \varepsilon\right)\right] \\
&\quad = \mathbb{E}\left[2\left(h_D(x) - \mathbb{E}[h_D(x)]\right)\left(\mathbb{E}[h_D(x)] - g(x)\right) - 2\left(h_D(x) - \mathbb{E}[h_D(x)]\right)\varepsilon\right] \\
&\quad = \mathbb{E}\left[2\left(h_D(x) - \mathbb{E}[h_D(x)]\right)\left(\mathbb{E}[h_D(x)] - g(x)\right)\right] \\
&\quad = \left(\mathbb{E}[h_D(x)] - g(x)\right)\mathbb{E}\left[2\left(h_D(x) - \mathbb{E}[h_D(x)]\right)\right] \\
&\quad = 0
\end{aligned}
$$

Here, the third equality follows from the zero mean of $\varepsilon$.

# Ensemble Methods vs Bias/Variance

Suppose that we could somehow sample $m$ independent datasets: $D_1, \ldots, D_m$.

We train $m$ models $h_{D_1}, \ldots, h_{D_m}$ and consider the ensemble model $h_{ens}(x) = \frac{1}{m} \sum_{\ell=1}^{m} h_{D_\ell}(x)$.

Does the ensemble $h_{ens}$ have a different bias-variance decomp.?

- The noise variance $\sigma^2$ does not change.
- The Bias does not change ($\mathbb{E}[h_{D_\ell}(x_k)]$ is independent of $\ell$):

$$\mathbb{E}\left[\frac{1}{m}\sum_{\ell=1}^{m} h_{D_\ell}(x_k)\right] - g(x_k) = \mathbb{E}[h_{D_\ell}(x_k)] - g(x_k)$$

- Only the Variance changes, it is *reduced*:

$$\mathbb{E}\left(h_{ens}(x_k) - \mathbb{E}[h_{ens}(x_k)]\right)^2 = \frac{1}{m}\mathbb{E}(h_{D_\ell}(x_k) - \mathbb{E}[h_{D_\ell}(x_k)])^2$$

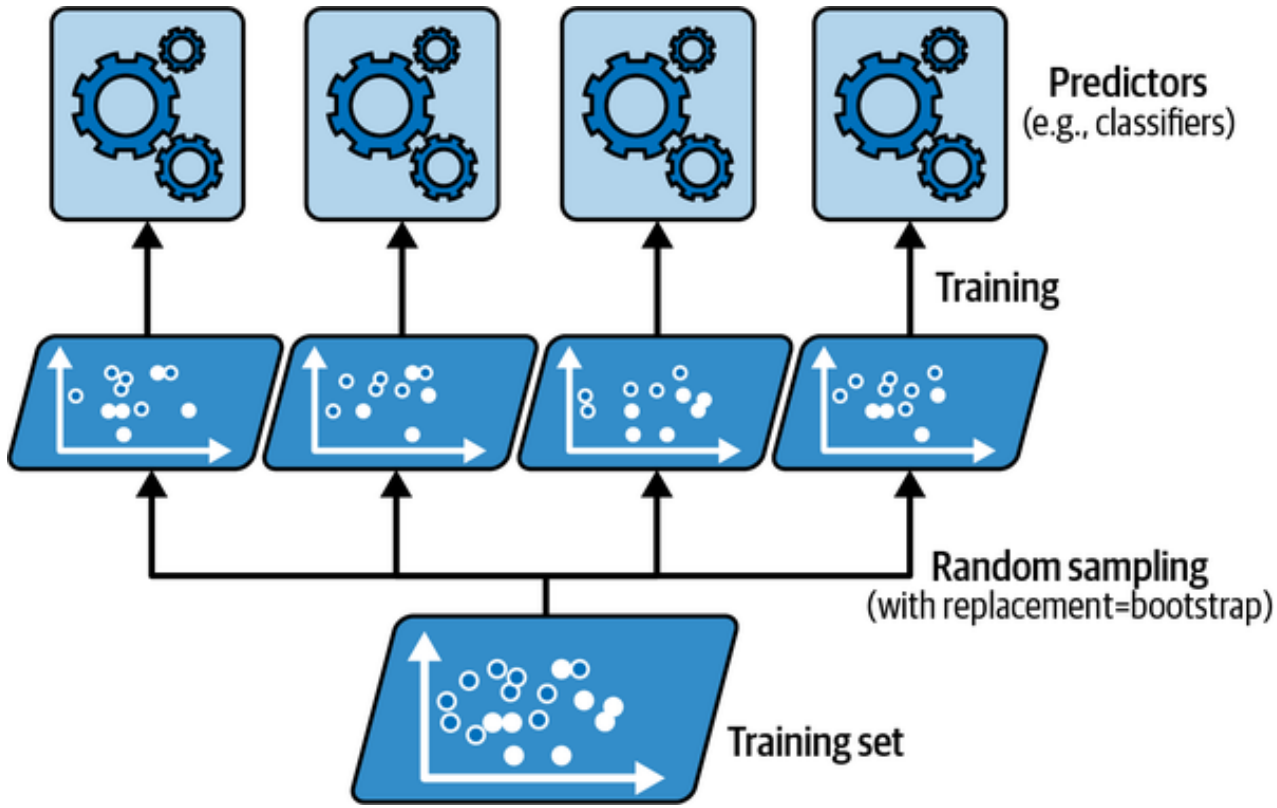
  The equality follows from basic facts about the variance of sums of independent variables.

Now, having $m$ independently sampled datasets is a real luxury!

# Bagging

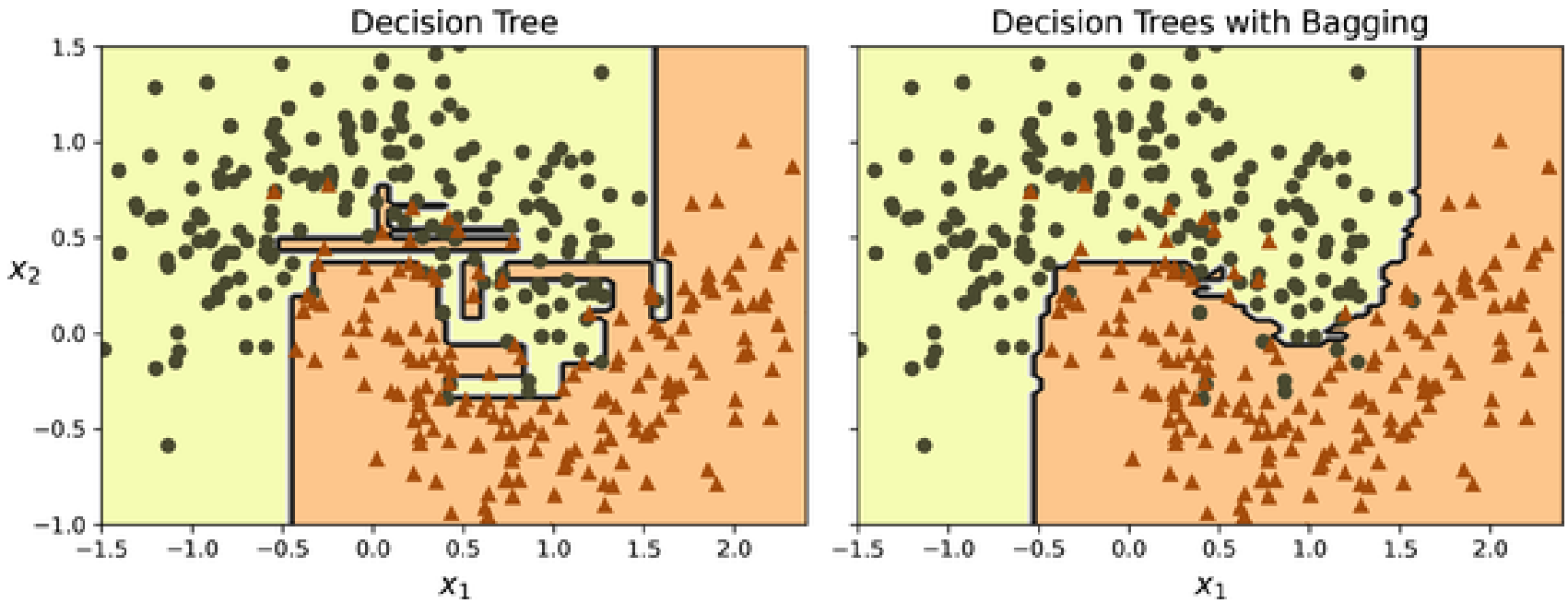
In practice, we use sampled subsets of a given dataset.

One example of using different subsets of the training set is *bagging (Bootstrap Aggregating)*.




- Bootstrap sampling = sampling data subsets with replacement
- Aggregating = majority voting (classification) or averaging (regression)

# Bagging Decision Trees




Left: A single decision tree (overfit)

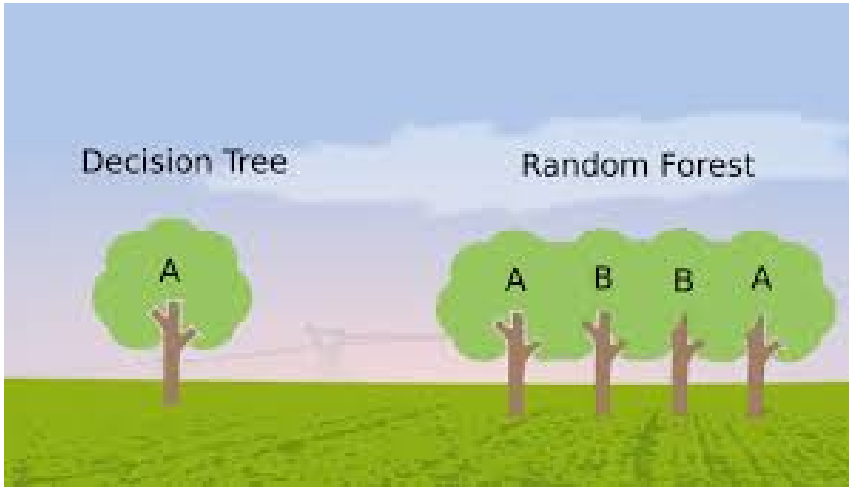
Right: A bagging ensemble of 500 trees

# Summary of Bagging

- Reduces overfitting (variance)
- Can work with any type of classifier
- Easy to parallelize

  Just train each model independently, inference can also be parallelized.
- Loses (some) interpretability even for interpretable models (how could you read 500 decision trees?)

# Random Forest




Random forest = bagging ensemble of decision trees.

Has all hyperparameters of decision trees + the number of trees

The random forest algorithm of scikit-learn introduces the following randomness:
When searching for a split attribute, look only into a randomly sampled subset of attributes (by default $\sqrt{n}$ of $n$ attributes).

This gives a greater diversity of the forests.

The notion of the feature importance can be easily generalized to random forests (computed over all trees).

# Boosting

An ensemble method in which weak learners are trained sequentially.

Each new learner tries to correct the error of the current ensemble.

There are several variants of boosting. We just have a look at the basic idea behind *AdaBoost* (adaptive boosting).

To implement this kind of learning algorithm, we need to be able to train models on weighted datasets.

# Weighted Training

We are going to use weights on samples.

Note that these are different from the weights used in neural-like models!

- Linear regression: Assume a given dataset $D = \{(\vec{x}_1, f_1), \ldots, (\vec{x}_p, f_p)\}$ and weights $a_1, \ldots, a_p$ associated to examples from $D$.

  Let $\vec{w}$ be a vector of model weights. We minimize the *weighted mean squared error*

$$E(\vec{w}) = \frac{1}{p} \sum_{i=1}^{p} a_i (\vec{w} \cdot x_i - f_i)^2$$

- Decision trees: Given dataset $D$ with $p$ samples, we have weights $a_1, \ldots, a_p \in \mathbb{R}$ (one weight for each example in $D$).

  Given a class $c \in C$, we compute the class probability $p_c$ as

$$p_c = \sum \{a_i \mid i\text{-th sample belongs to } c\} \,/\, \sum_{i=1}^{p} a_i$$

  Everything else (Gini impurity, etc.) is the same.

# AdaBoost Idea

- Train classifiers using weighted samples in the training dataset.
- Start with uniform weights, that is, each sample in the dataset has the same weight.
- In $k$-th iteration:
  - train a new classifier $h_k$ on weighted samples,
  - obtain a coefficient $\alpha_k > 0$ of the classifier $h_k$,
  - Consider the current ensemble classifier

$$h^k_{ens}(x) = \text{sign}\left(\sum_{\ell=1}^{k} \alpha_\ell h_\ell(x)\right)$$

Shift the weights so that the "most wrong" training instances for the current ensemble have the largest weights.
Consider a training example $(x, c)$ with $c \in \{-1, 1\}$. Then $h^k_{ens}$ misclassifies $x$ iff $c \cdot \sum_{\ell=1}^{k} \alpha_\ell h_\ell(x) < 0$. The more negative this number is, the more wrong the ensemble classifier is.

After $K$ iterations, the ensemble classifier $h^K_{ens}$ is the output.

For details, see more advanced courses in Machine Learning.

# Summary of Ensemble Methods

- Ensemble methods may improve "independently" trained models by grouping them and letting them decide collectively.
- Technically, the ensembling decreases the model variance.
- There are several approaches to ensembling:
  - Bagging - training of several models on bootstrap subsamples of the training dataset (random forest is an example)
  - Boosting - ensemble is built sequentially, the newly added model possibly solves the hardest instances
  - There are many more algorithms (gradient boosting, etc.)

# THE END