Machine Learning

Basics - 2

Thang Vu

7.11.2025

Outline

- Models
- Learning
- Features

Outline

- Models
- Learning
- Features

Different Classifiers: Examples

- Parametric classifiers
  - Logistic regression
  - Support vector machines (SVM)
  - Neural networks (NN)
- Non-parametric classifiers
  - K-nearest neighbors
  - Decision trees

Different Classifiers: Examples

- Parametric classifiers
  - Logistic regression
  - Support vector machines (SVM)
  - Neural networks (NN)
- Non-parametric classifiers
  - K-nearest neighbors
  - Decision trees

Support Vector Machines

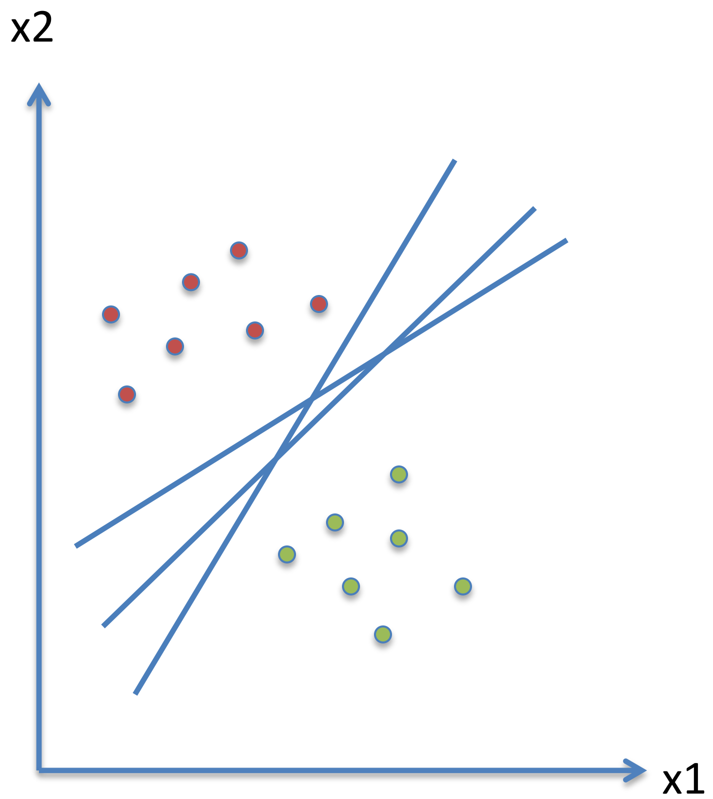
- A supervised learning method for both classification and regression
- Key ideas:
  - Maximize the margin between classes
  - Support vectors
    - Identify support vectors to form the optimal hyperplane
  - Kernel trick
    - Convert a non-linear problem to a linear problem in a new transformed space

Support Vector Machines

- A supervised learning method for both classification and regression
- Key ideas:
  - Maximize the margin between classes
  - Support vectors
    - Identify support vectors to form the optimal hyperplane
  - Kernel trick
    - Convert a non-linear problem to a linear problem in a new transformed space

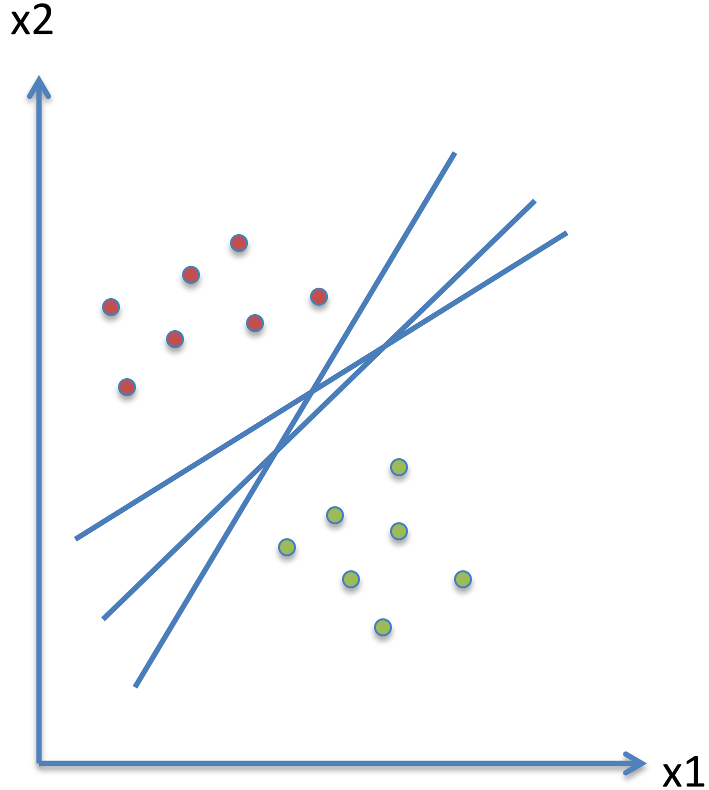
Linear SVM

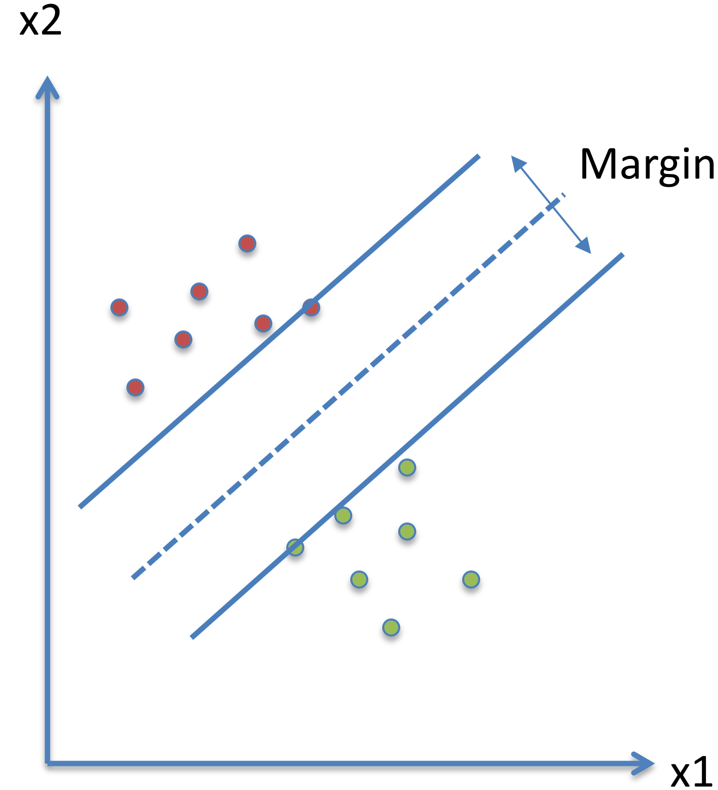
- For classification
- Goal: Search for the best separation hyperplane



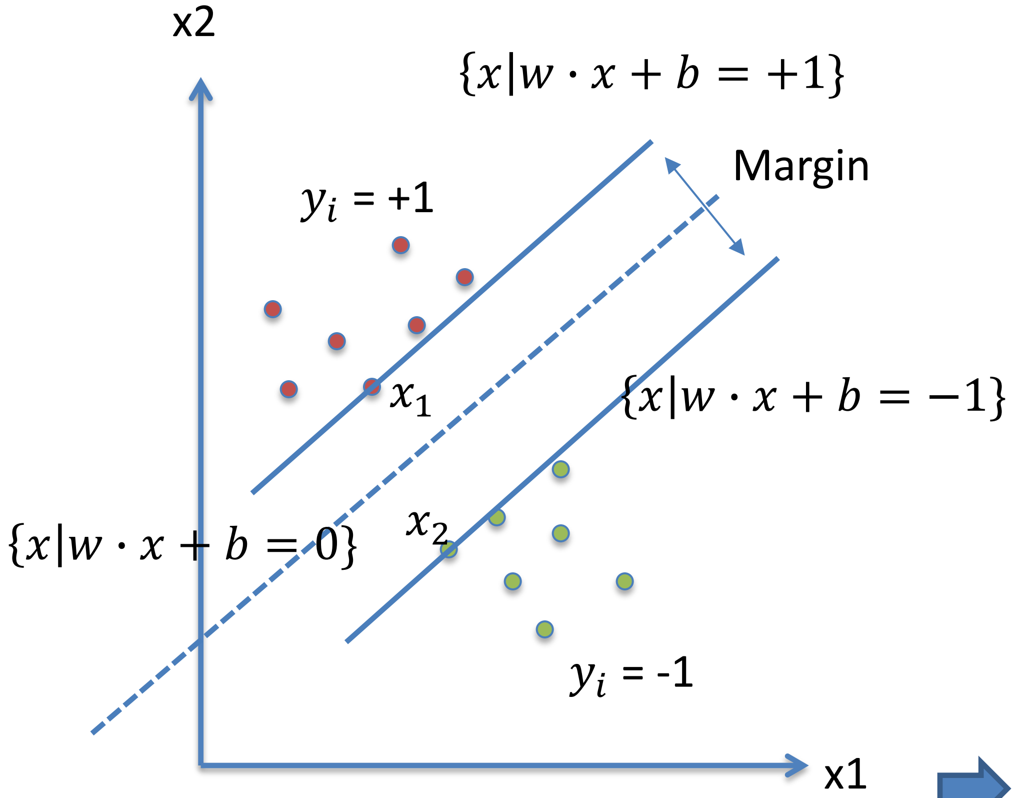
Linear SVM

- For classification
- Goal: Search for the best separation hyperplane
- Idea: Maximize the margin ➡ improve generalization





Optional: Linear SVM



$$Distance \quad \frac{w \cdot (x_1 - x_2)}{||\vec{w}||}$$

$$(w \cdot x_1) + b = +1$$
$$(w \cdot x_2) + b = -1$$

$$\Rightarrow \quad w(x_1 - x_2) = 2$$

$$\Rightarrow \quad \frac{w}{||w||}(x_1 - x_2) = \frac{2}{||w||}$$

Optional: Linear SVM

To find the best hyperplane:

- Maximize $\frac{2}{||\vec{w}||}$ ⇒ minimize $\frac{1}{2}||\vec{w}||^2$

- Subject to:

  $y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1 \; i = 1, \ldots, n$

- i.e., all the training data is correctly classified
  - n: number of training samples

Optional: Linear SVM – Lagrange Method

- How to solve an optimization problem with constraints?
  - Lagrange method:
    Find the sattel point of the following equation:

$$L_p = \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{n} \alpha_i (y_i(\vec{w} \cdot \vec{x}_i + b) - 1)$$

$$= \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{n} \alpha_i y_i(\vec{w} \cdot \vec{x}_i + b) + \sum_{i=1}^{n} \alpha_i$$

  - with $\alpha_i \geq 0$ being Lagrange multipliers

Optional: Linear SVM – Lagrange Method

- $$\frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i = 0$$

  $$\Rightarrow \quad \vec{w} = \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i$$

- $$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0$$

- **Lagrangian Dual Problem**: (Kuhn-Tucker theorem) Instead of *minimizing* over w, b subject to constraints with α, we can *maximize* over α subject to these relations for w and b!

Optional: Linear SVM – Lagrange Method

- Primal problem:

$$\min L_p = \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{n} \alpha_i y_i (\vec{w} \cdot \vec{x}_i + b) + \sum_{i=1}^{n} \alpha_i$$

$$\text{s.t.} \alpha_i \geq 0$$

- $$\vec{w} = \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i \qquad \sum_{i=1}^{n} \alpha_i y_i = 0$$

- Dual problem:

$$\max L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\vec{x}_i \cdot \vec{x}_j)$$

$$s.t. \sum_{i=1}^{n} \alpha_i y_i = 0 \ \& \ \alpha_i \geq 0$$

Support Vectors

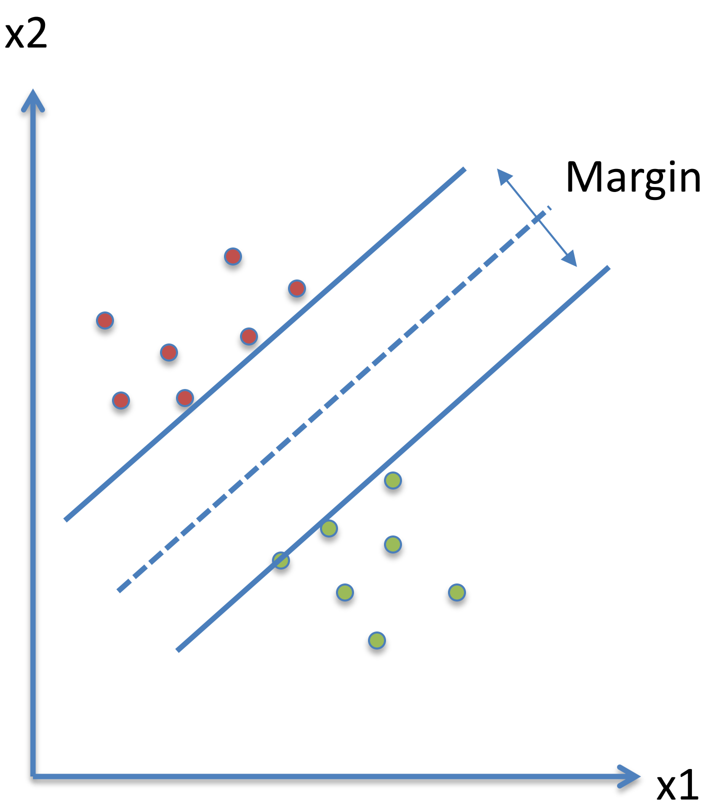
- In most cases, $\alpha_i = 0$
- If $\alpha_i > 0$, we have so called support vectors $\overrightarrow{x_i}$

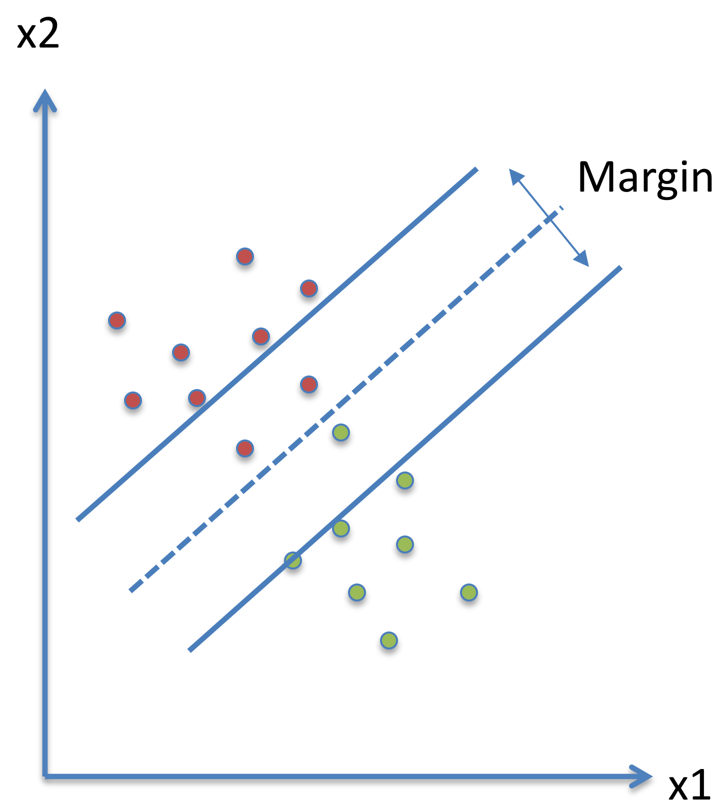
- $\vec{w}$ is a linear combination of support vectors

$$\vec{w} = \sum_{i=1}^{N} \alpha_i y_i \overrightarrow{x_i}$$

Soft Margin

- Idea:
  - Allow also misclassification during training
  - Improve generalization





Linear SVM with Soft Margin

To find the best hyperplane:

- Minimize $\frac{1}{2}||\vec{w}||^2 + C(\sum_{i=1}^{n} \xi_i)^p$
- With relaxed conditions:
  $y_i(w \cdot \overrightarrow{x_i} + b) \geq 1 - \xi_i \quad i = 1, \ldots, n \; and \; \xi_i > 0$

- C > 0 is a penalty to errors
  - Large C means less misclassification
  - Small C allows large margin

Nonlinear Classification

- Idea:
  - Transform the training data $\in R^n$ into another space R^m in which the problem is linearly separable
  - Then apply linear SVM to solve it
  - Usually $m > n$
- Example: Monom transformation

$$\phi\colon\ R^2 \longrightarrow R^3$$
$$(x_1, x_2) \longmapsto (x_1^2, \sqrt{2}x_1 x_2, x_2^2)$$

Nonlinear Classification

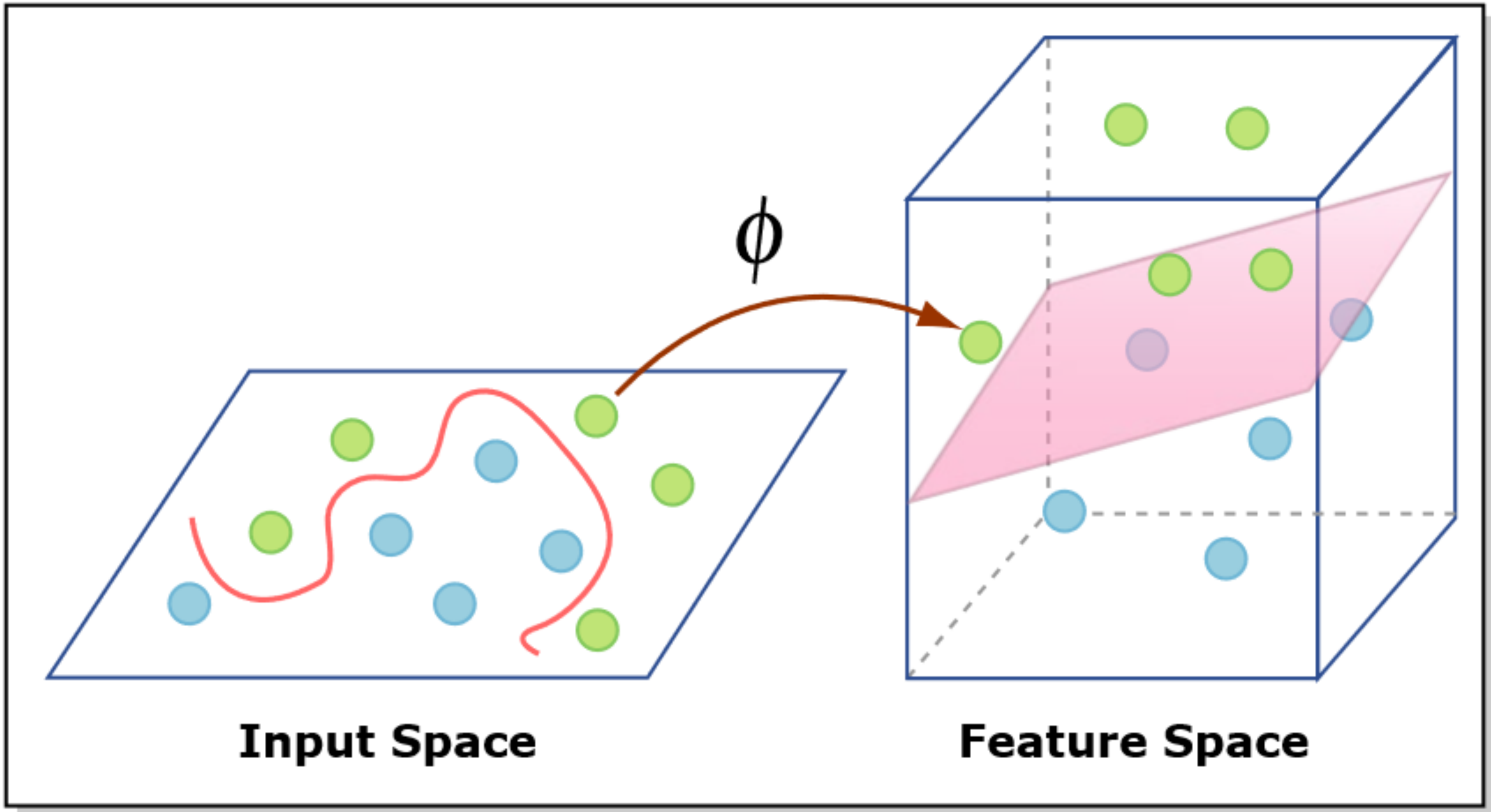


Image by MIT OpenCourseWare.

Optional: Kernel Trick

- Problem:
  Tranformation of data into a higher-dimensional space and computation there can be expensive
- But: remember:

$$\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j$$

- => All we do with x_i and x_j is computing the dot product
- Kernel trick: If our transformation is a kernel, we have: K(x,y) = ϕ(x) · ϕ(y)

Optional: Kernel Trick

- Kernel trick: If our transformation is a kernel, we have: K(x,y) = ϕ(x) · ϕ(y) => it defines similarity in our higher-dimensional feature space!
- So, the function we optimize now is:

$$\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\vec{x}_i \cdot \vec{x}_j)$$

- => Very efficient

Kernel Functions: Examples

- Scalarproduct:

$$K(x, y) = x \cdot y$$

- Polynomial:

$$K(x, y) = (x \cdot y + c)^d$$

- Radial Basis Function (RBF)

$$K(x, y) = e^{-||x-y||^2 / 2\sigma^2}$$

Multiclass SVM

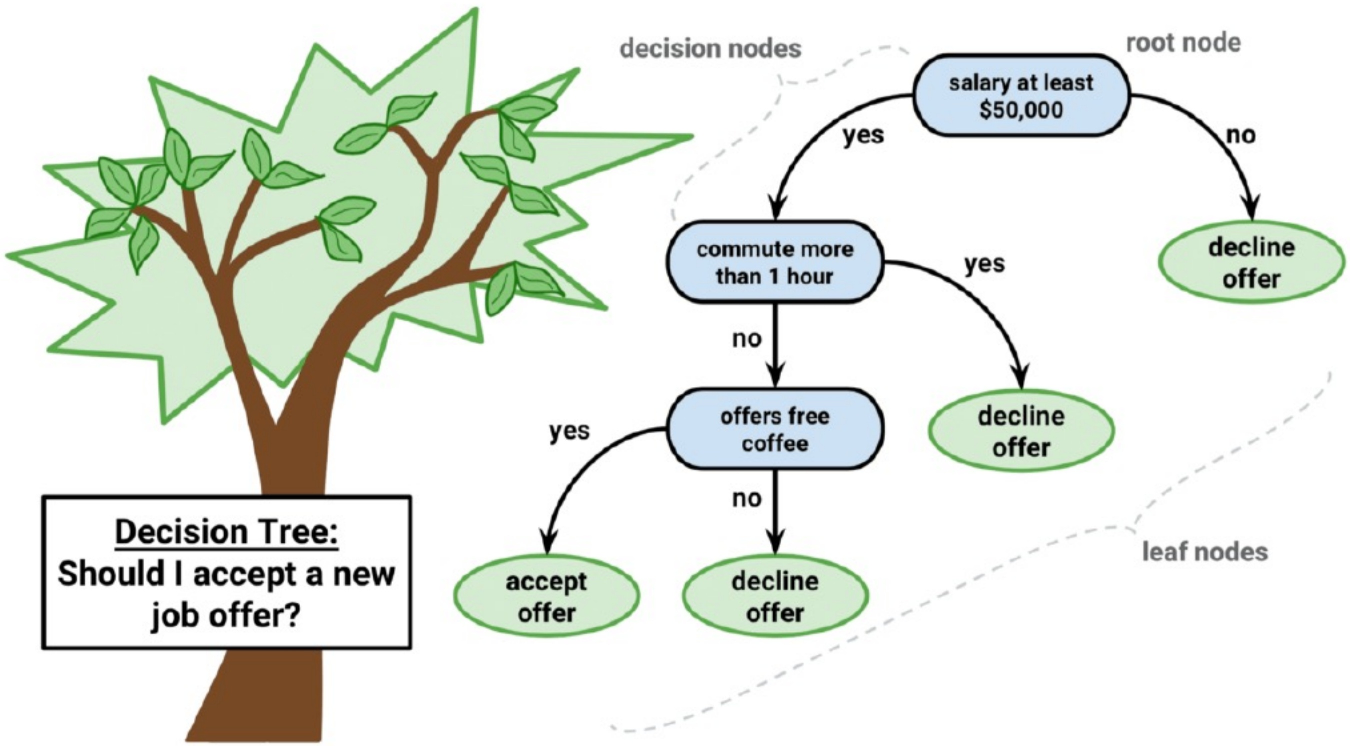
- Ideas:
  - Decompose multiclass classification problem into multiple binary classification problems
  - Use the major voting mechanism to predict the target
- Methods:
  - One vs. rest
    - N binary classifiers
    - Class with the maximum score wins
  - One vs. one
    - $\binom{N}{2}$ binary classifiers
    - Class with maximum number of votes wins

Live Voting

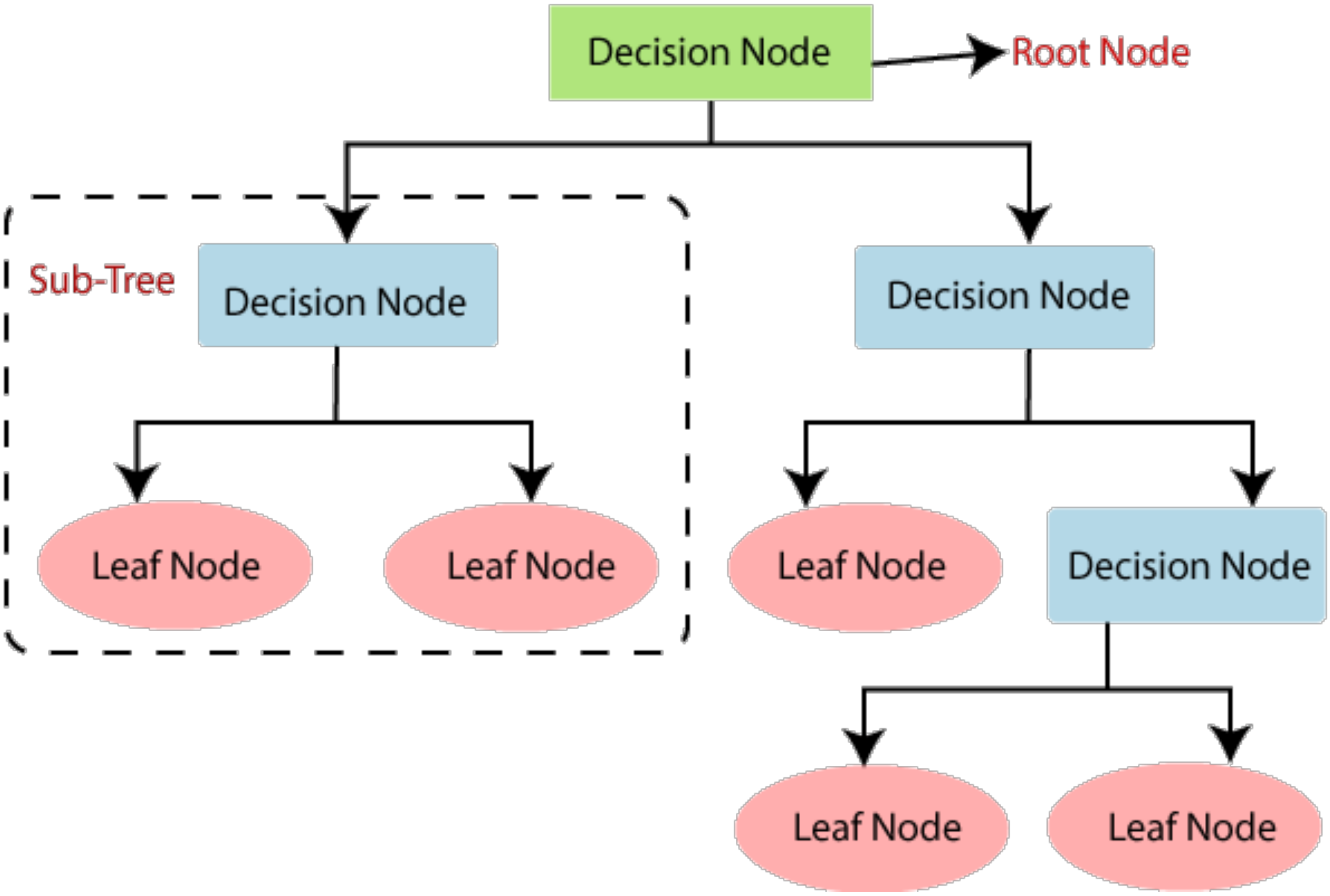
Different Classifiers: Examples

- Parametric classifiers
  - Logistic regression
  - Support vector machines (SVM)
  - Neural networks (NN)
- Non-parametric classifiers
  - K-nearest neighbors
  - Decision trees

A Decision Tree



A Decision Tree



How to Construct

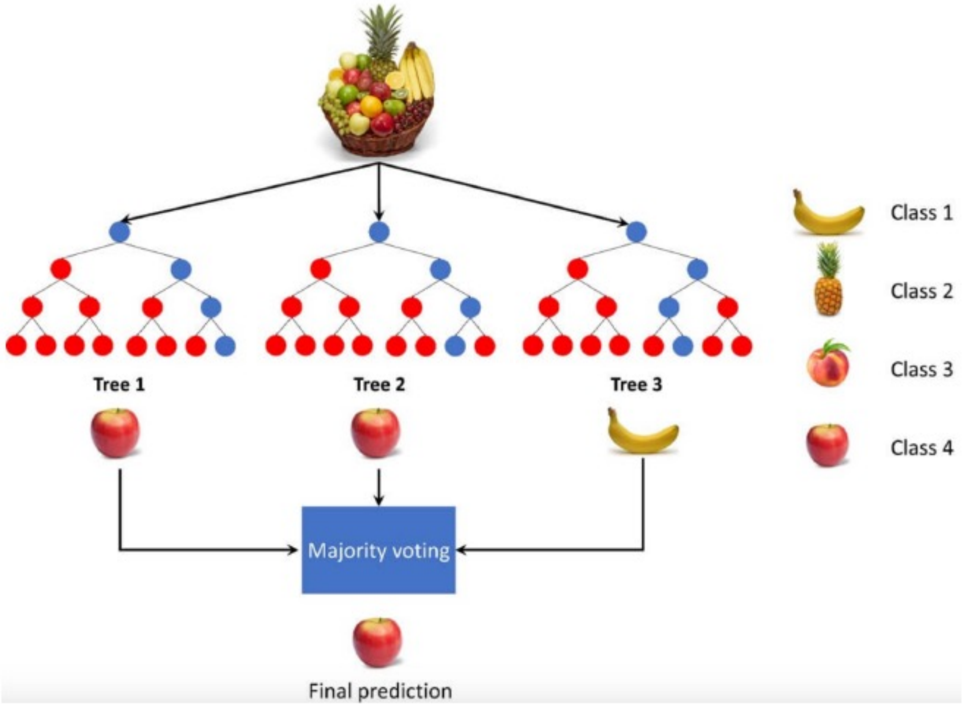
- Features are used for splitting
  - Categorical, e.g., yes or no
  - Continuous features, e.g., larger or smaller than a value
- But which features?
  - Build the tree using a hierarchical approach, starting from the root node
  - Maximize the information gain
- When to stop?
  - Maximal tree depth (hyperparameter)
  - Maximal number of features (hyperparameter)

Pros and Cons

- Pros:
  - Easy to interpret and visualize
  - Can handle both numerical and categorical data (regression and classification)
  - Can handle both linear and non-linear data
  - Features do not need to be scaled or normalized
- Cons:
  - Risk of overfitting
  - Risk of bias toward frequent classes

Random Forest

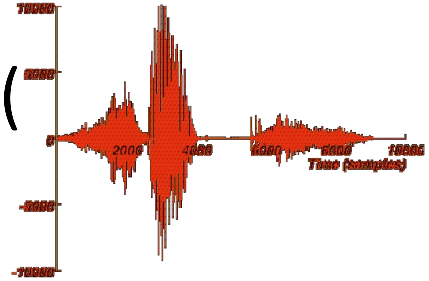
- A combination of several decision trees
- 'Random':
  - Each tree uses only a subset of the features
  - Each tree has access to a subset of the training data



Live Voting

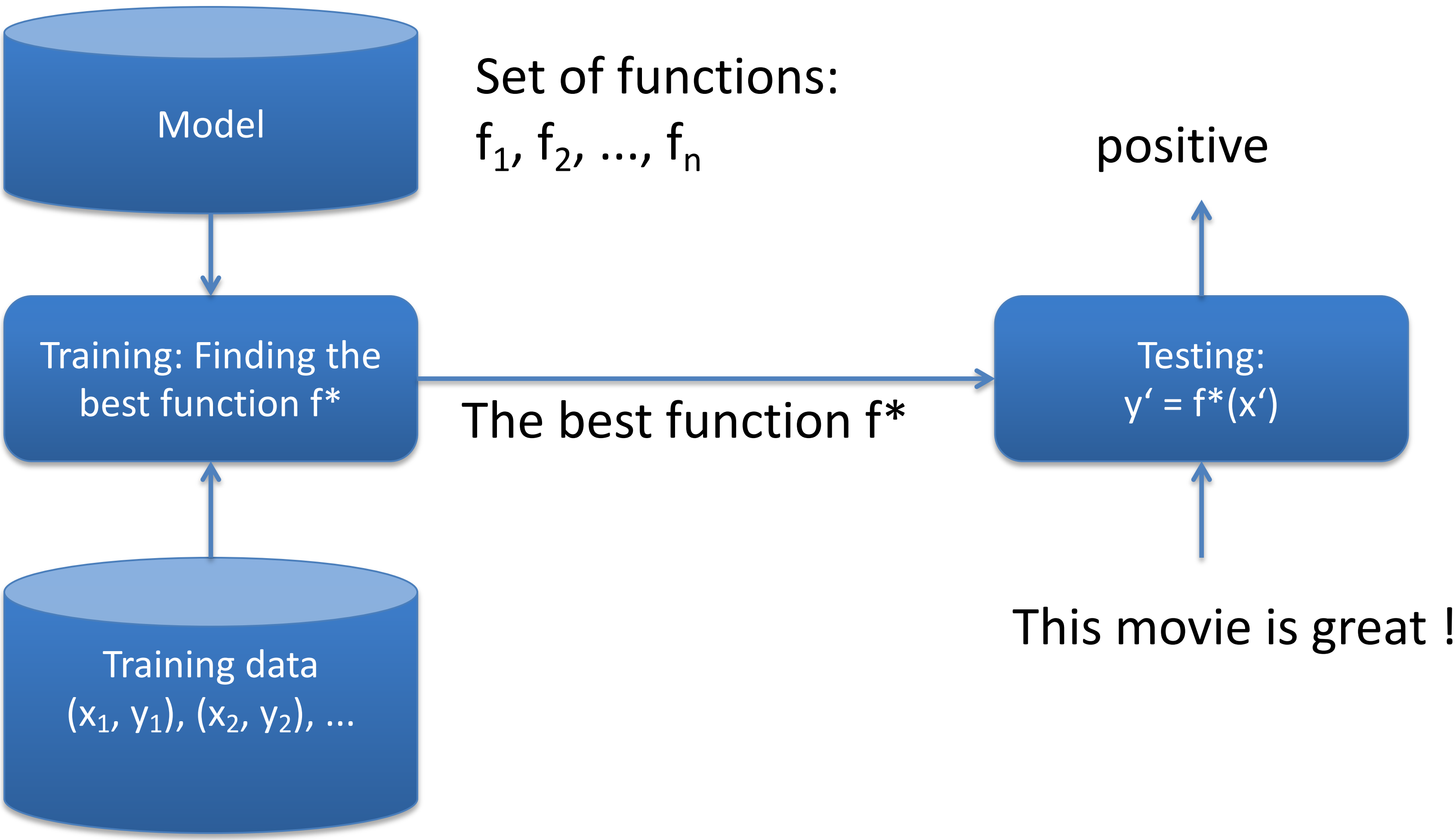
Learning Parametric Models

Learning ≈ Searching for a Function

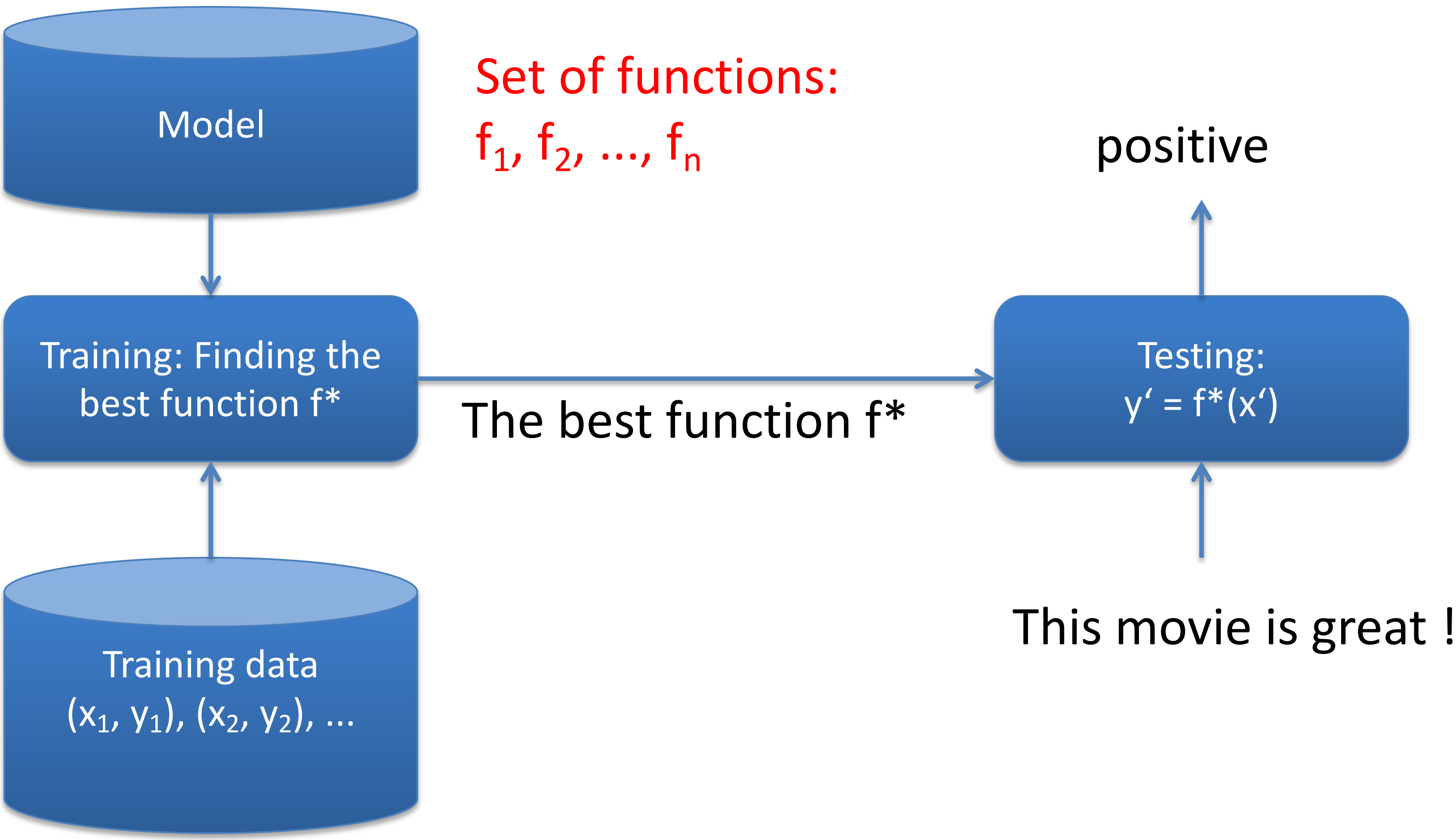
- Image classification:
  - f () = motorbike
- Phoneme recognition:
  - f (

) = /t/ /u/
- Natural language processing:
  - f (*This book is great!*) = positive

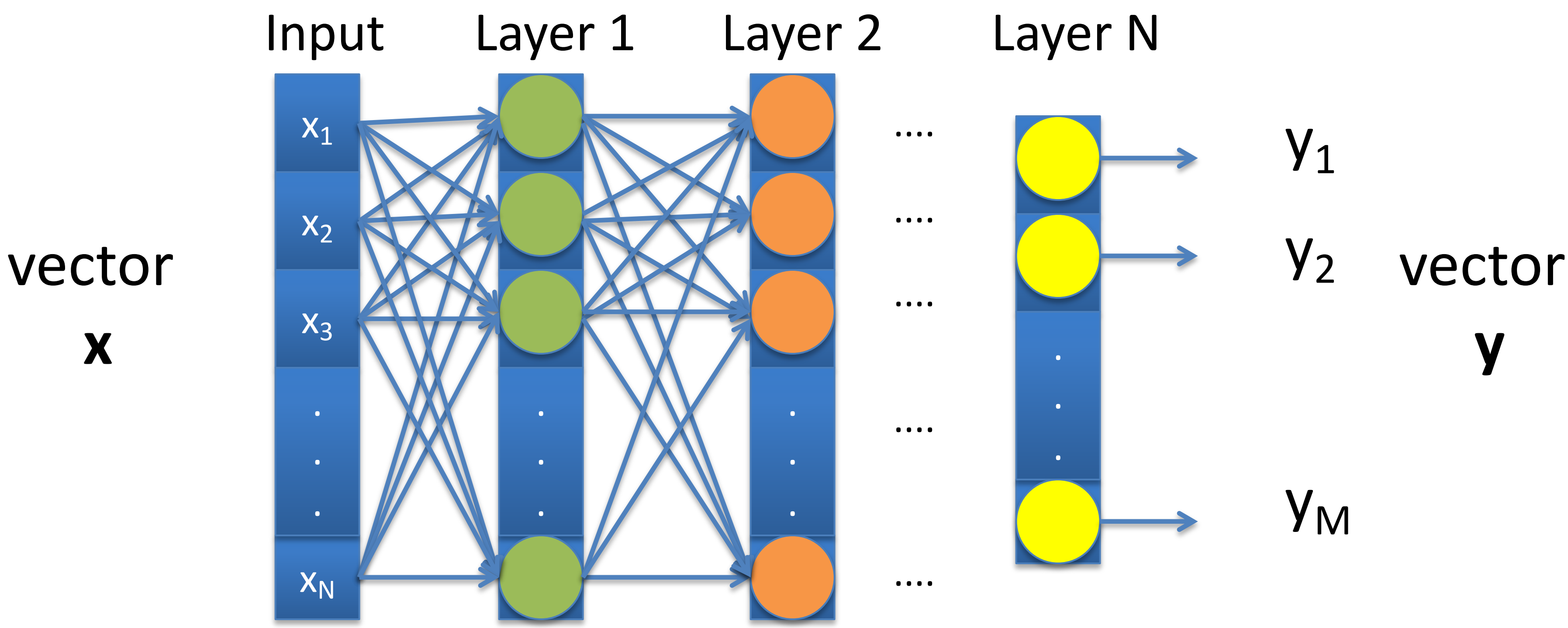
Supervised Learning Framework



Supervised Learning Framework

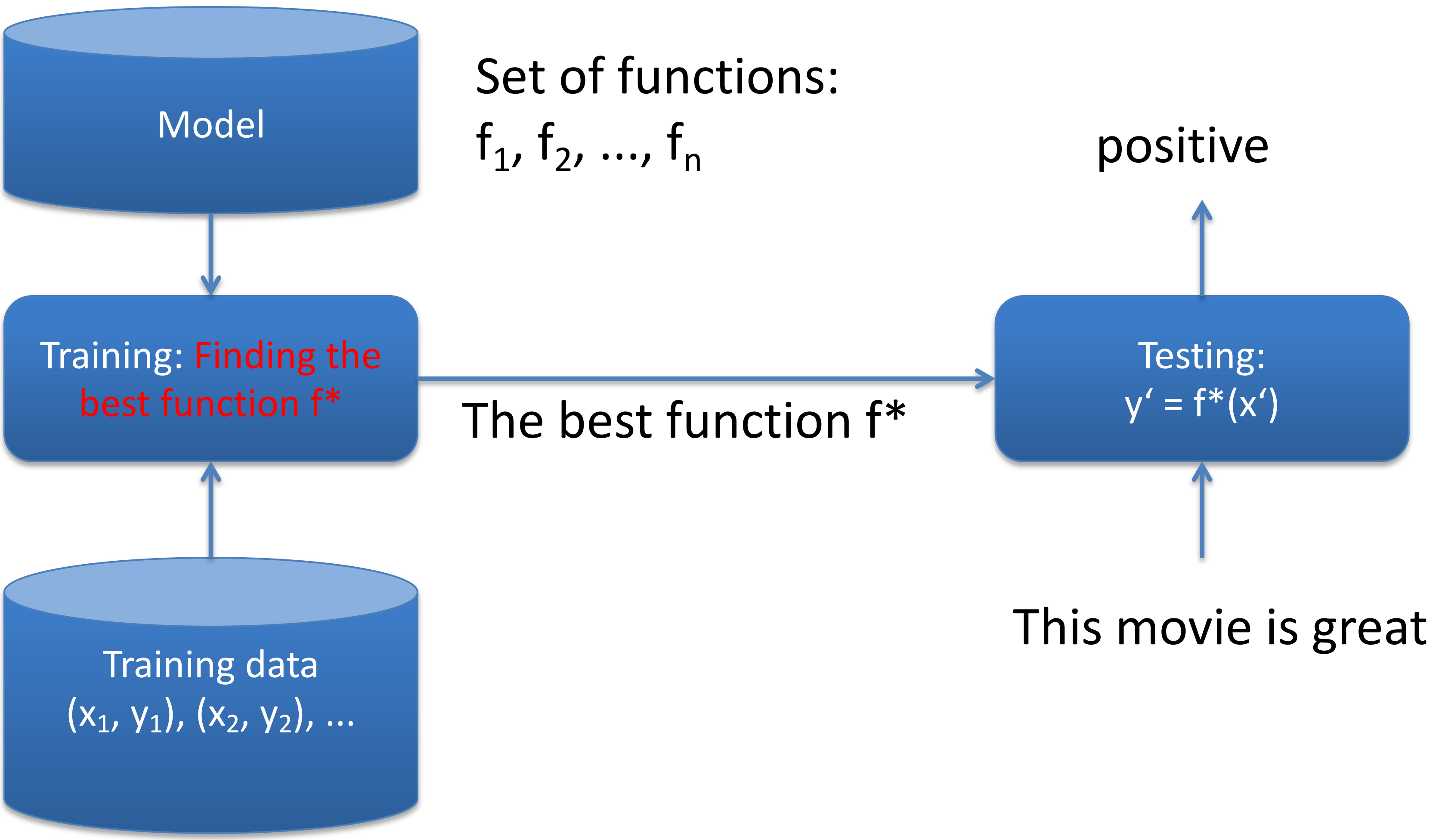


Functions of Neural networks



$$y = f(x) = \sigma(W^L \ldots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \ldots + b^L)$$

Supervised Learning Framework



Which function is the *best* function?

- Best function means *best* parameters

$$y = f(x) = \sigma(W^L \ldots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \ldots + b^L)$$

- The function itself depends on the parameter set

$$f(x) = f(x, \theta)$$

$$\theta = \left\{ W^1, b^1, W^2, b^2, \ldots, W^L, b^L \right\}$$

- Seach the *best* function f*

→ Search the *best* parameter set $\theta *$

Loss or cost function

- Define a function for the trainable parameters C(θ)
  - C evaluates how bad a parameter set is
  - The best parameter is the one that minimizes C(θ)

$$\theta^* = \arg\min_{\theta} C(\theta)$$

  - C(θ) is called loss function

Empirical risk minimization

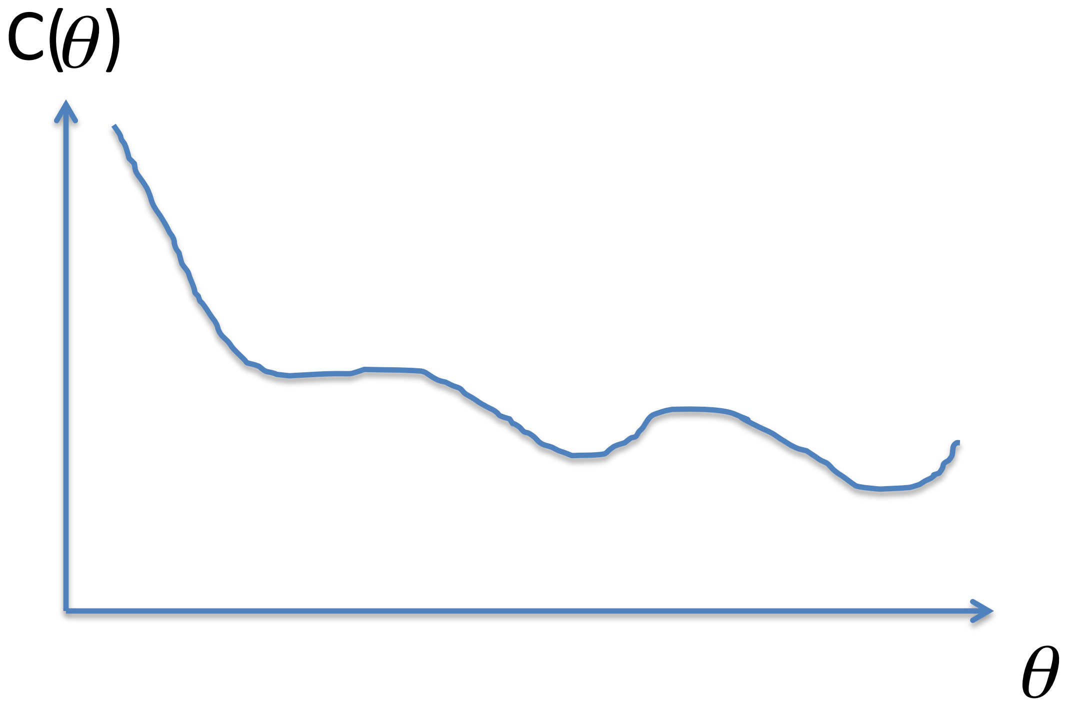
- Given a finite set of training data
- Empirical risk = average loss on this training data

$$C(\theta) = \frac{1}{|D|} \sum_{(x,y)} c(f(x), y)$$

$$= \frac{1}{|D|} \sum_{(x,y)} c(\theta)$$

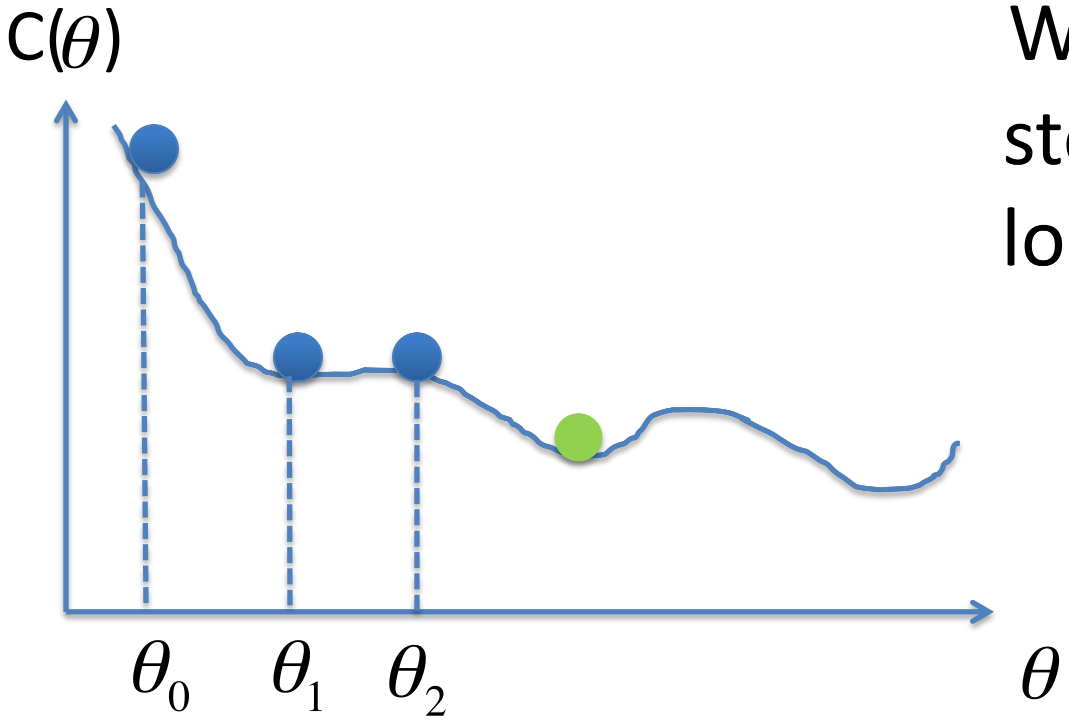
One Demonstrative Example of the Loss

- First consider that θ has only one variable



Gradient descent

- First consider that θ has only one variable

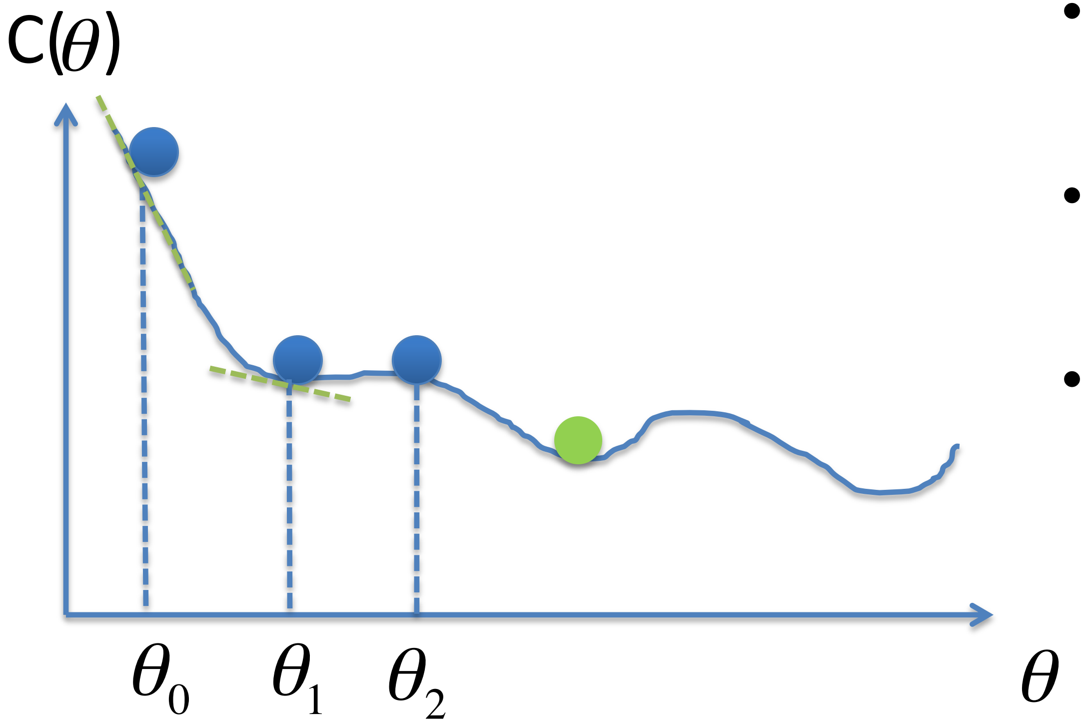


Start at somewhere ...

When the process stops, we will get the local minima

Gradient descent

- First consider that θ has only one variable



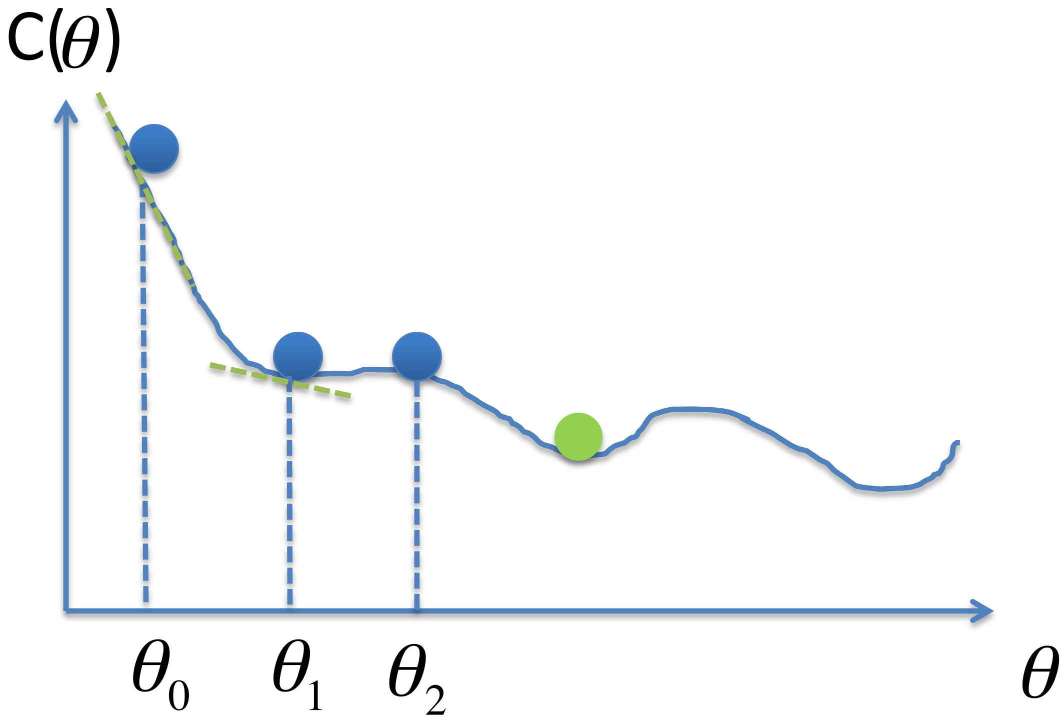
- Randomly start at θ_0
- Compute $dC(\theta_0)/d\theta$

  $\theta_1 \leftarrow \theta_0 - \eta dC(\theta_0)/d\theta$
- Compute $dC(\theta_1)/d\theta$

  $\theta_2 \leftarrow \theta_1 - \eta dC(\theta_1)/d\theta$
- ……

Gradient descent

- First consider that θ has only one variable



- Randomly start at θ_0
- Compute dC(θ_0)/d θ

$$\theta_1 \leftarrow \theta_0 - \eta dC(\theta_0)/d\theta$$

- Compute dC(θ_1)/d θ

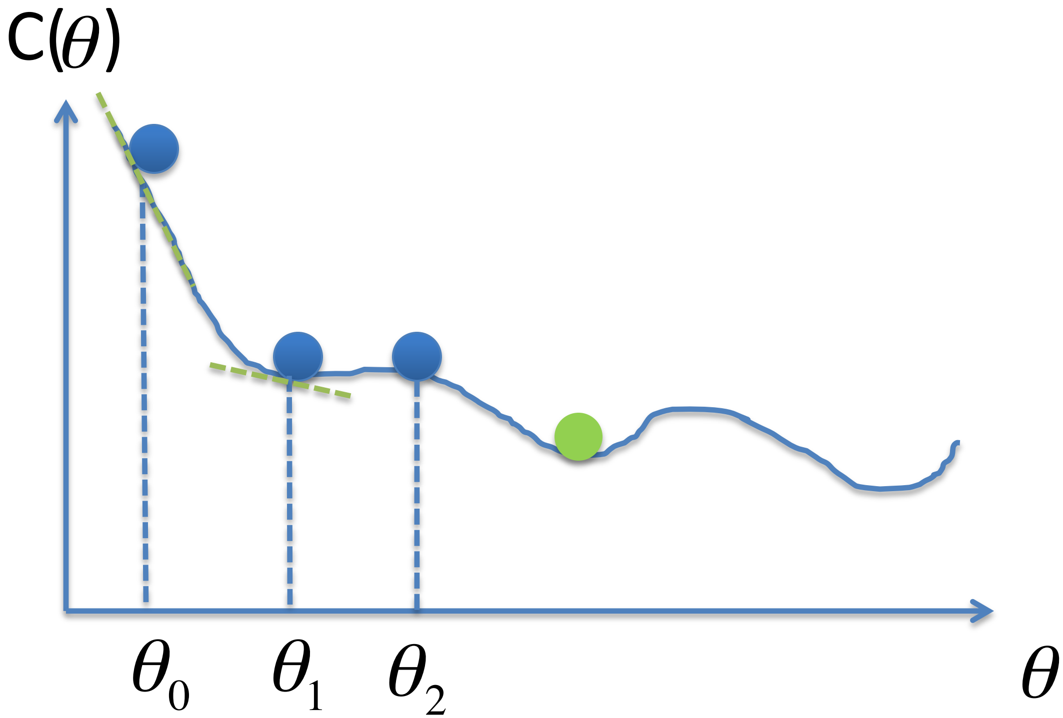
$$\theta_2 \leftarrow \theta_1 - \eta dC(\theta_1)/d\theta$$

- ……

the gradients

Gradient descent

- First consider that θ has only one variable



- Randomly start at θ_0
- Compute $dC(\theta_0)/d\theta$

$$\theta_1 \leftarrow \theta_0 - \eta dC(\theta_0)/d\theta$$

- Compute $dC(\theta_1)/d\theta$

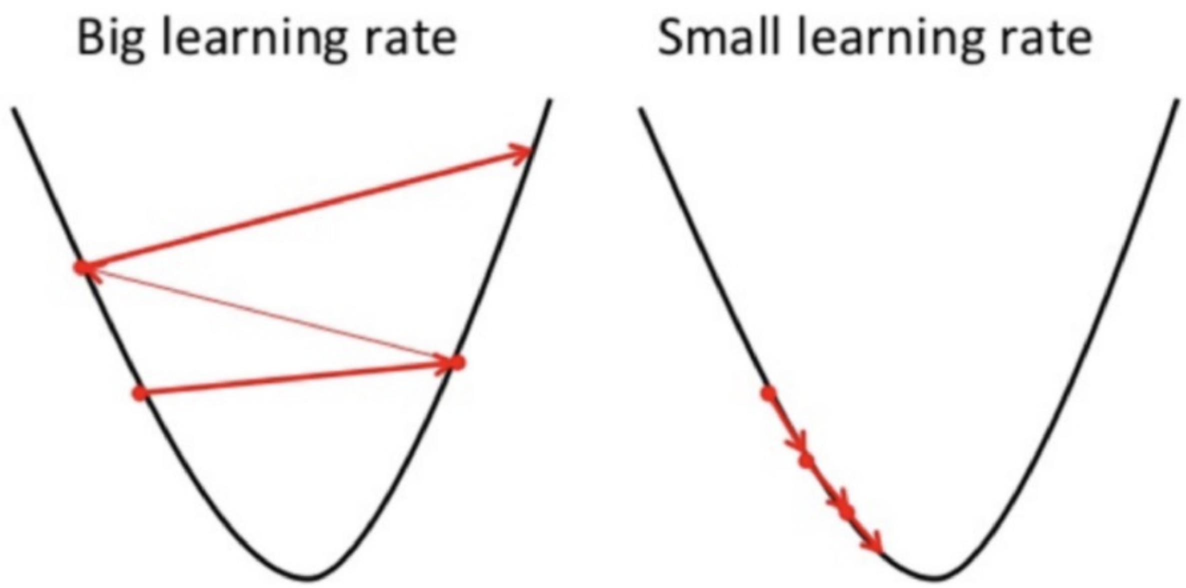
$$\theta_2 \leftarrow \theta_1 - \eta dC(\theta_1)/d\theta$$

- ……

Learning rate

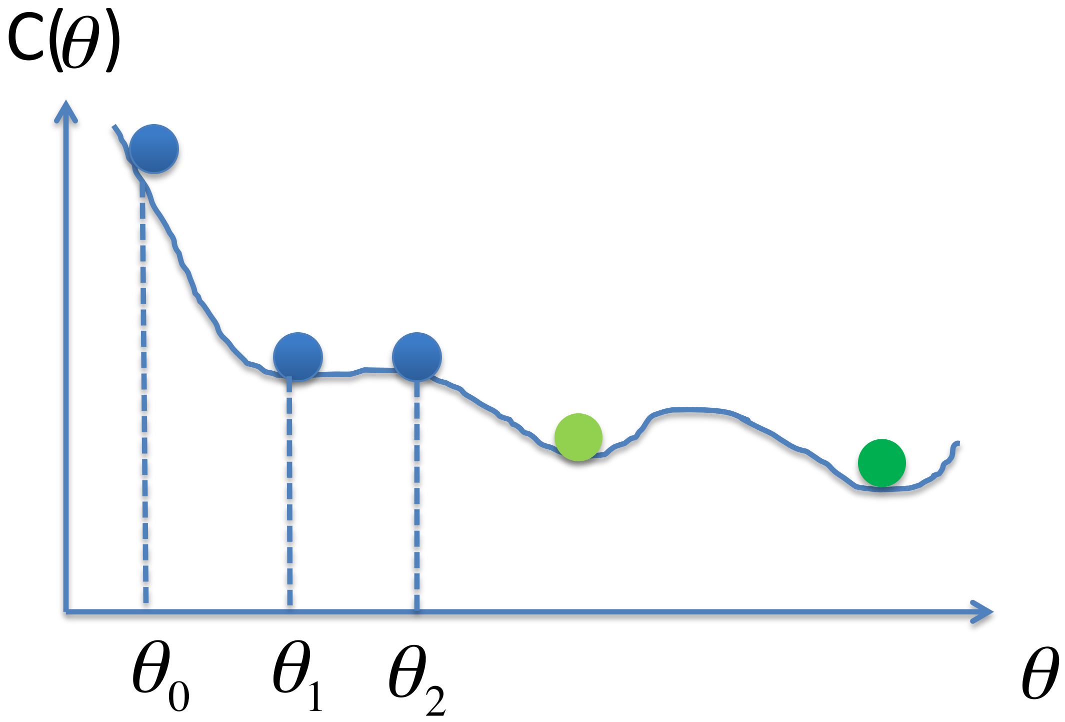
Learning rate

- Small learning rate ➡ extensive training time
- Big learning rate ➡ divergence



Gradient descent

Gradient descent can only help to find local minima, NOT the global minimum



Gradient descent

Now assume that θ has two variables

- Randomly start at $\theta_0 = \left[\theta_0^1, \theta_0^2\right]$
- Compute the gradients of $C(\theta\ )$ at θ_0

  $\nabla C(\theta_0) = \left[\partial\theta_0^1 / \partial\theta^1, \partial\theta_0^2 / \partial\theta^2\right]$
- Update the parameters:

  $$\theta_1^1 = \theta_0^1 - \eta \partial C(\theta_0^1) / \partial\theta^1$$
  $$\theta_1^2 = \theta_0^2 - \eta \partial C(\theta_0^2) / \partial\theta^2$$
  $$\rightarrow \quad \theta_1 = \theta_0 - \eta \nabla C(\theta_0)$$
- Compute the gradients of $C(\theta\ )$ at θ_1

  $\nabla C(\theta_1) = \left[\partial\theta_1^1 / \partial\theta^1, \partial\theta_1^2 / \partial\theta^2\right]$
- ……

Live Voting

In Practice

Build a Machine Learning Model

1. A task
2. Preparing datasets
3. A model
4. A cost/loss function
5. An optimization procedure

Build a Machine Learning Model

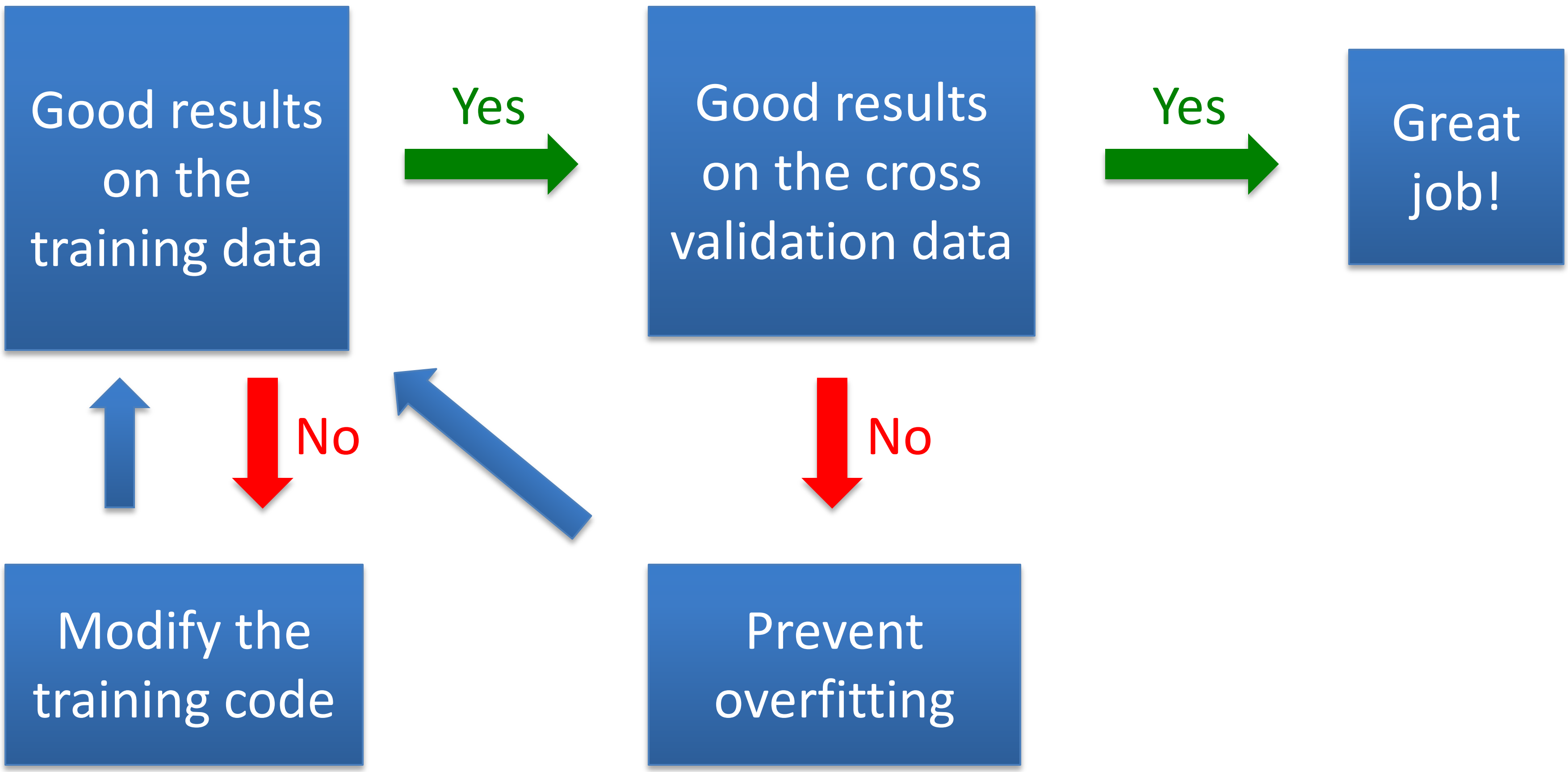
1. A task
2. Preparing datasets
3. A model
4. A cost/loss function
5. An optimization procedure
6. Output analysis
7. Deployment

Learning Recipe

- Split the data in three parts

Training Data	Validation Data	Evaluation Data
Used for training and monitoring the performance	Used for monitoring the performance and tuning hyperparameters	Don‘t touch it till the deadline

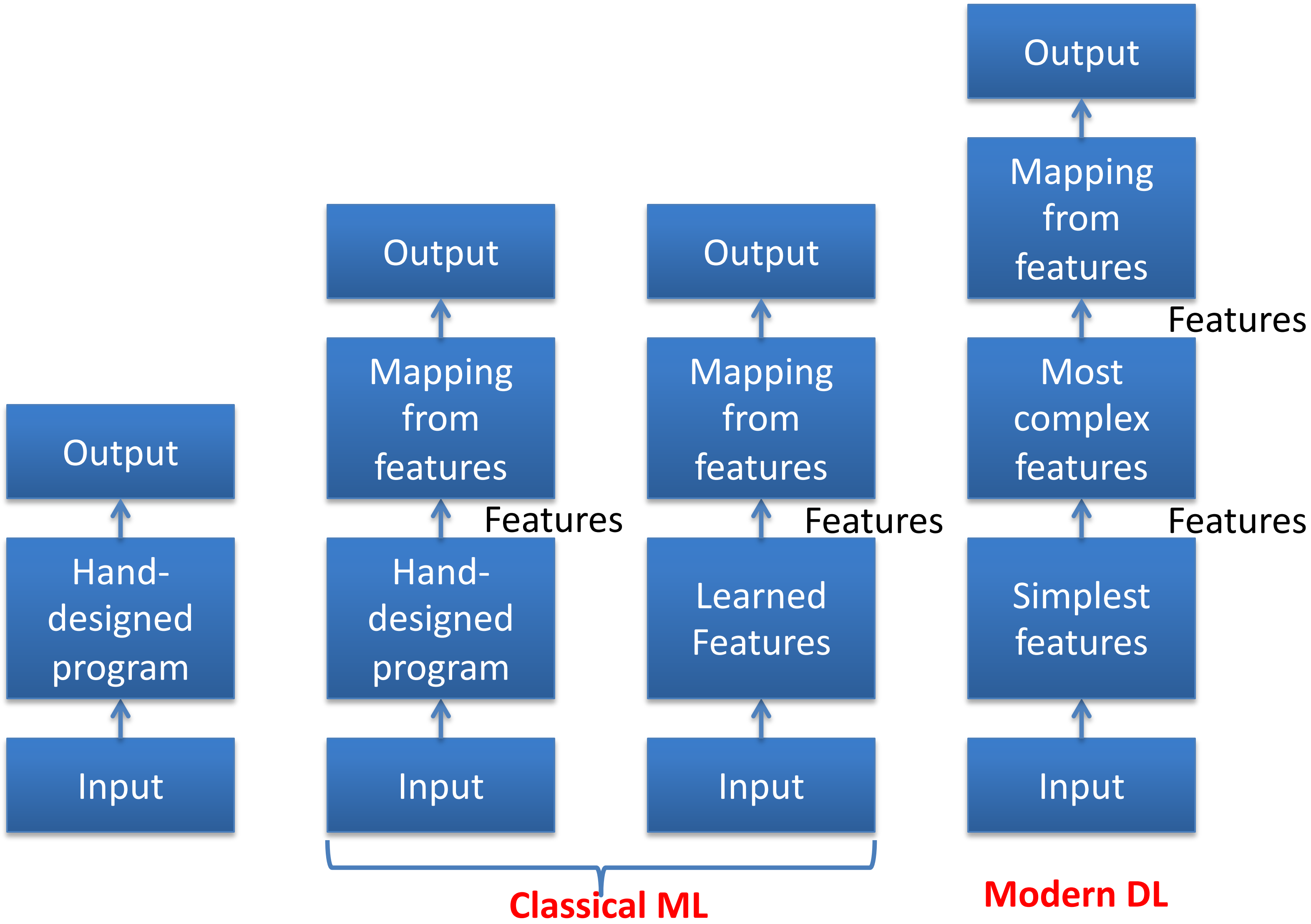
Learning Recipe



Outline

- Models
- Learning
- Features

Machine Learning Pipeline



The Curse Of Dimensionality

- Many machine learning problems become exceedingly difficult when the number of dimensions of the feature space is high
- The number of possible configurations of features is much larger than the number of training examples
  - Think of k-nearest neighbors regression when testing on a new data point
  - If the feature space is large, it isn't easy to find training examples associated with this new data point

Feature Selection

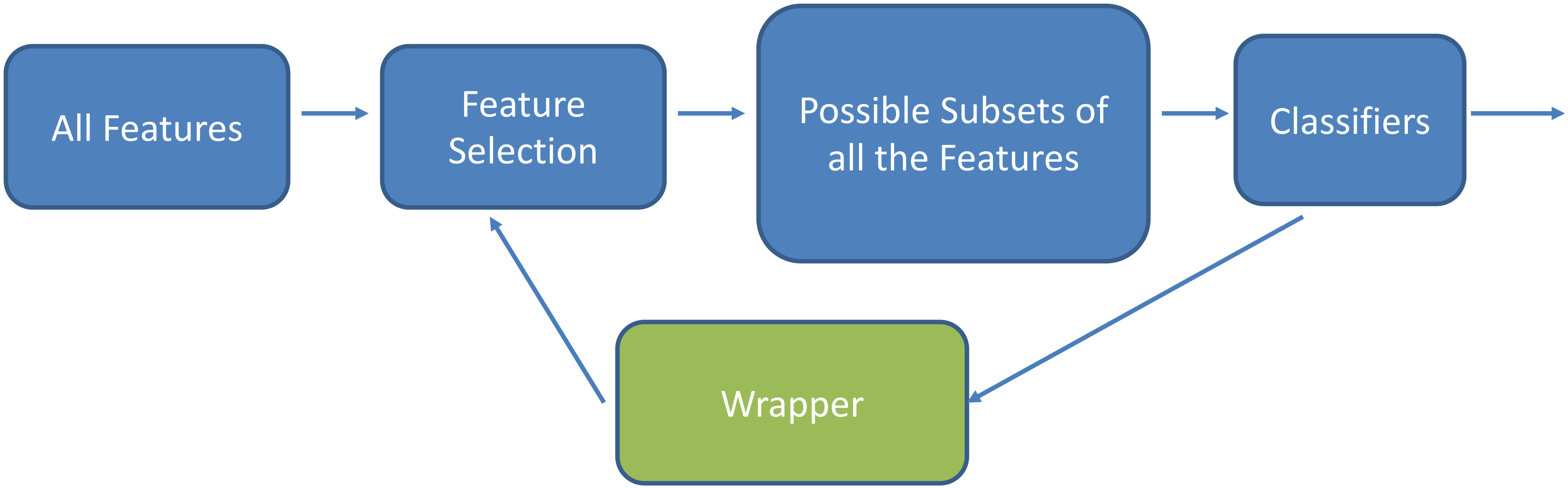
Feature Selection

- Select a subset of relevant features
- Motivation:
  - Features are redundant or irrelevant (noises)
- Advantages:
  - To avoid the curse of dimensionality
  - To simplify the models and make them more understandable
  - To shorten the training time
  - To improve the generalization of the models

Methods for Features Selection

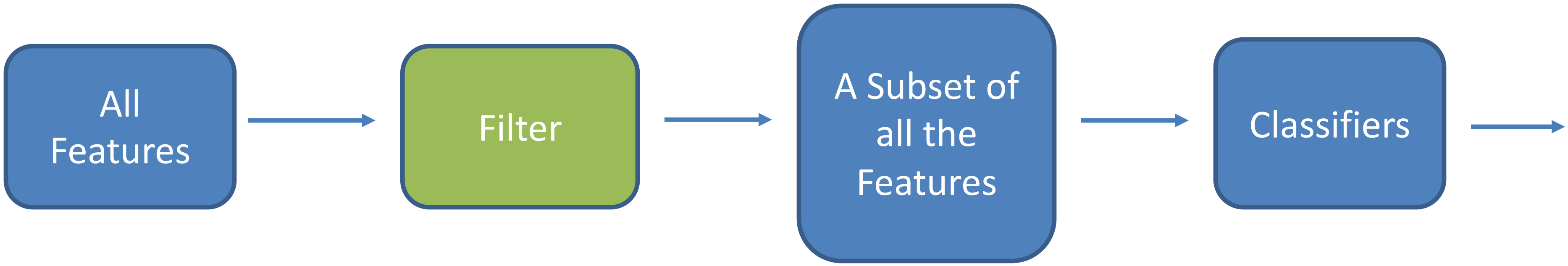
- Wrapper methods, e.g.
  - Greedy forward selection
  - Greedy backward elimination
- Filter methods, e.g.
  - Correlation based
  - Mutual information based
- Embedded methods, e.g.
  - Random forests
  - LASSO methods

Wrapper Methods



- Use the classifier to estimate the relevance of the features
- Pros:
  - Selected features are optimal for classification
- Cons:
  - Time consuming
  - Overfitting problem

Filter Methods



- Evaluate the features independent from the classifiers
- Individual ranking based on statistical test
- Pros:
  - Robust against overfitting
- Cons:
  - It could miss the most effective features

Filter Methods - Examples

- Removing low variance features:
  - Remove all features whose variance doesn‘t meet some threshold
  - By default: Removes all zero-variance features, i.e., features that have the same value in all samples

Embedded Methods

- Combine wrapper and filter methods
- Integration of feature selection as a part of the model construction, e.g., LASSO methods
- LASSO methods:
  - introduce an additional penalty term based on the absolute values of the coefficients (L1-norm)
  - During training, irrelevant features will obtain a regression coefficient close to 0
  - In other words, features with positive regression coefficients are automatically selected

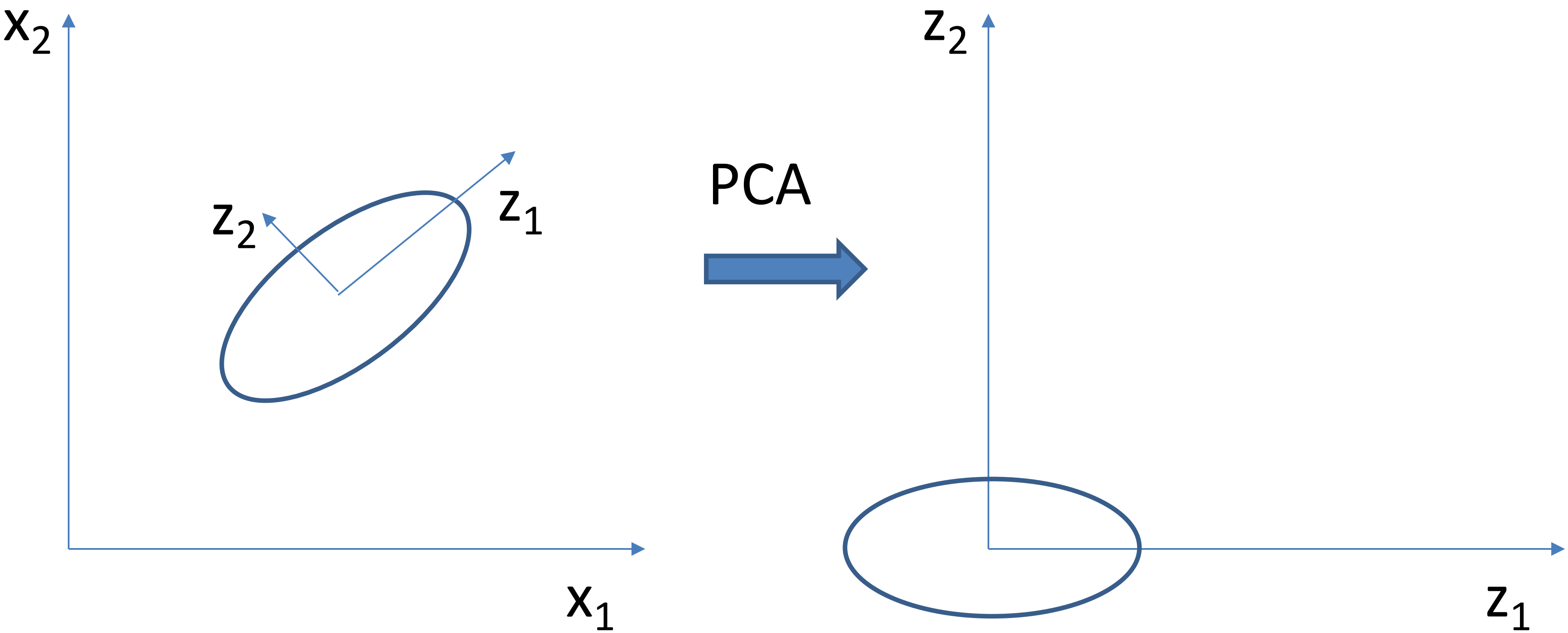
Feature Extraction or Construction

Feature Extraction or Construction

- Transform the feature space to satisfy a set of predefined criteria
- It is not necessarily the same as dimensionality reduction
  - It can be used for dimensionality reduction
- Two popular methods:
  - Principal component analysis (PCA)
  - Linear discriminative analysis (LDA)

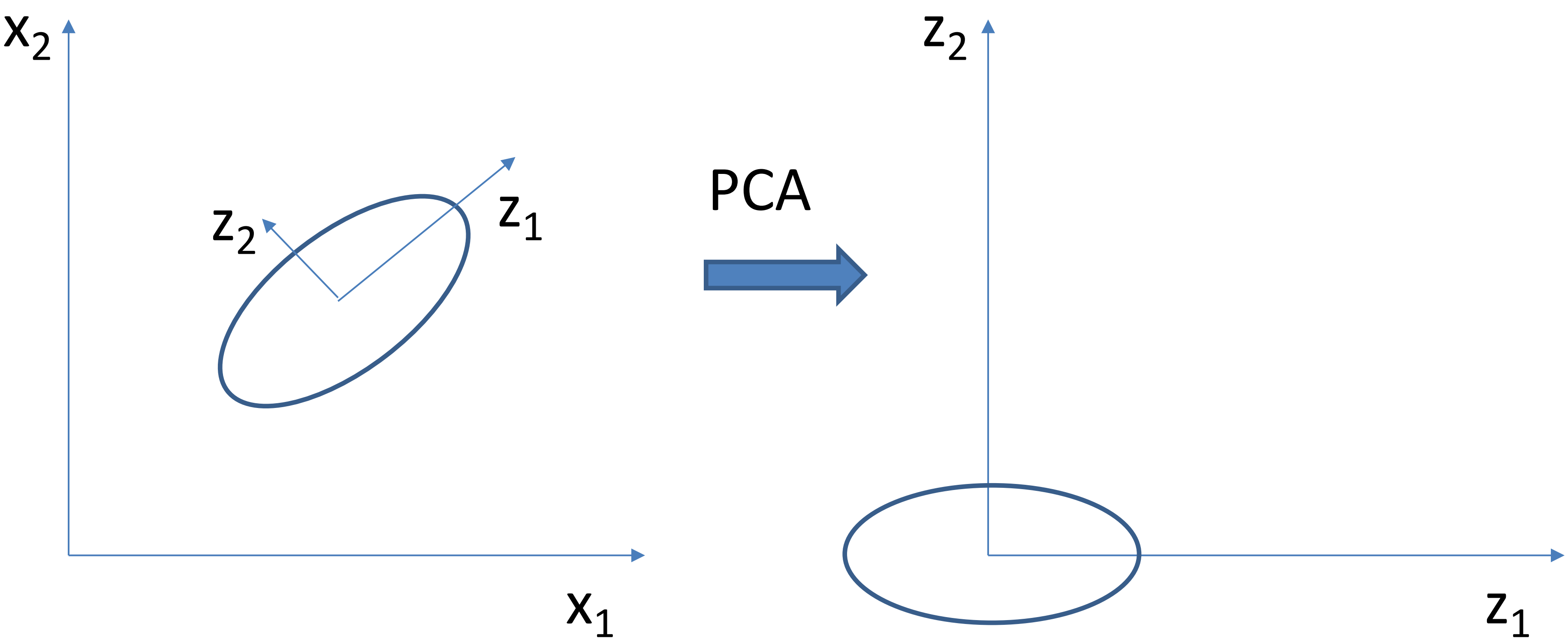
PCA

- Goal: A new space in which the transformation to this space causes the least information lost



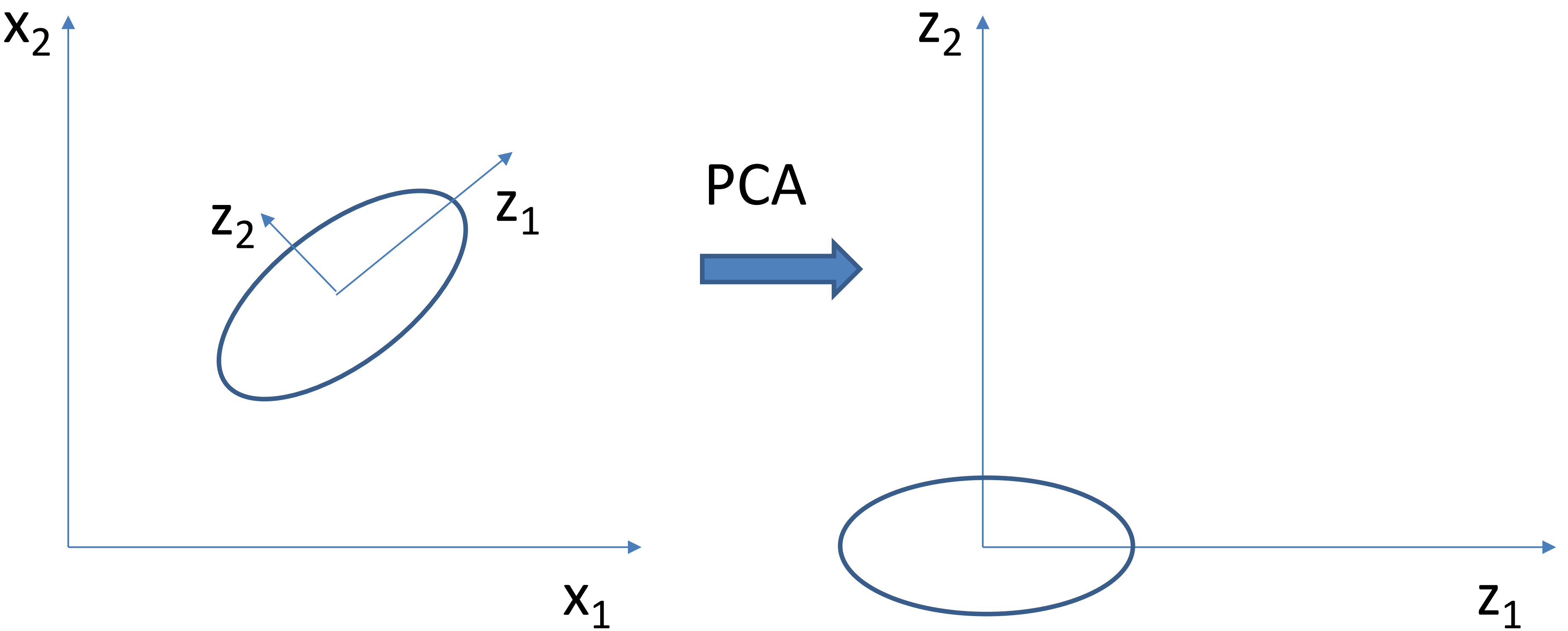
PCA

- Idea: learn a transformation so that the axes of the new space have the maximum variances



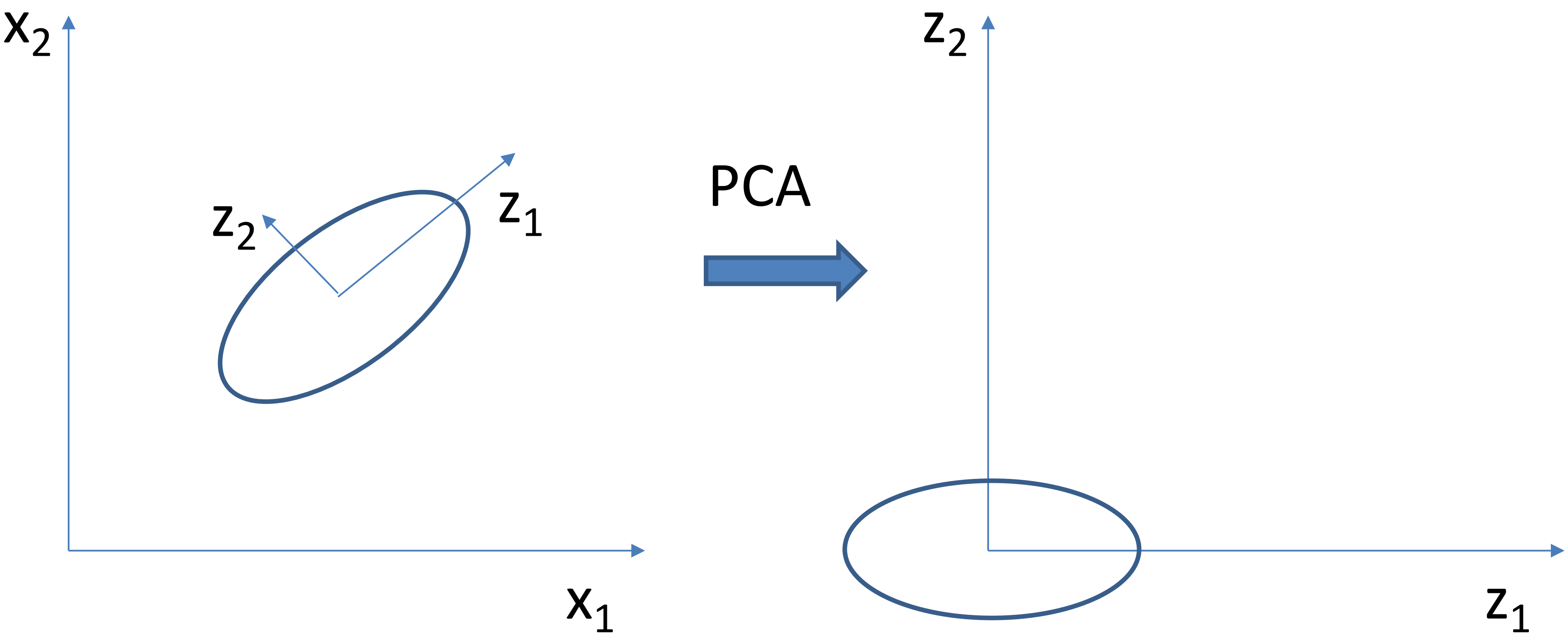
PCA

- Can be used to reduce the dimensionality by removing dimensions with smallest variances

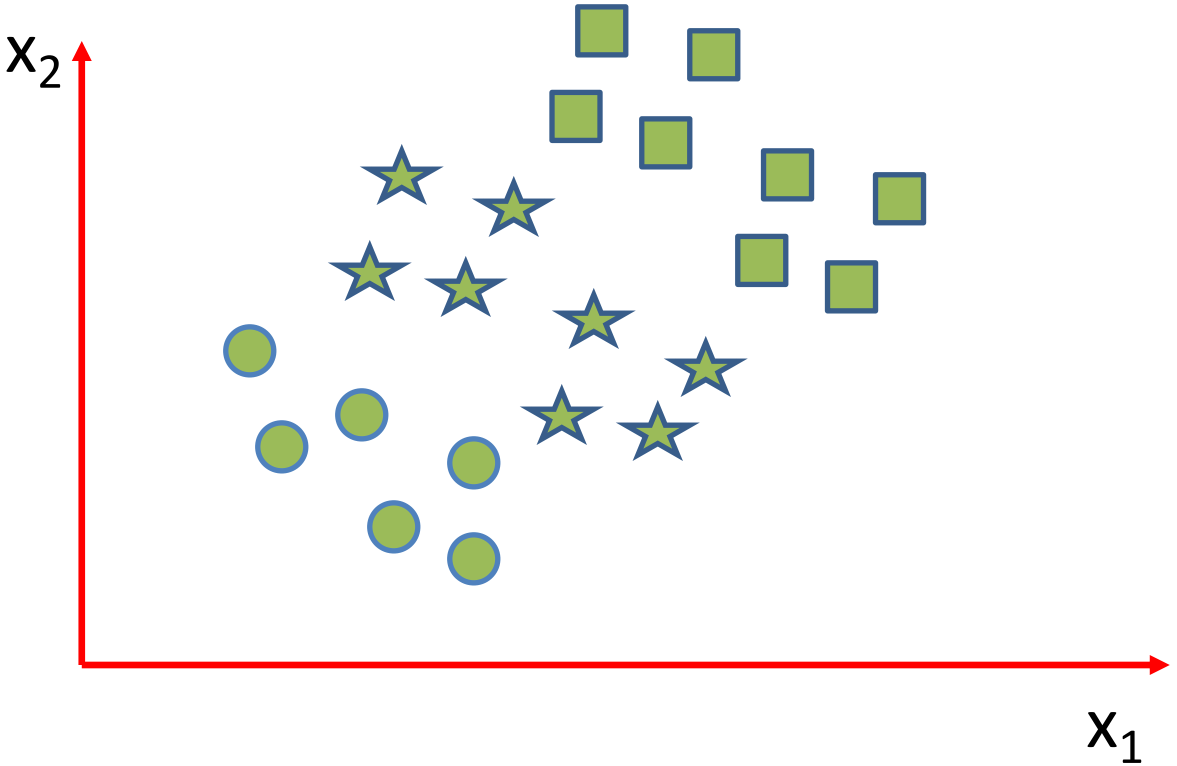


PCA

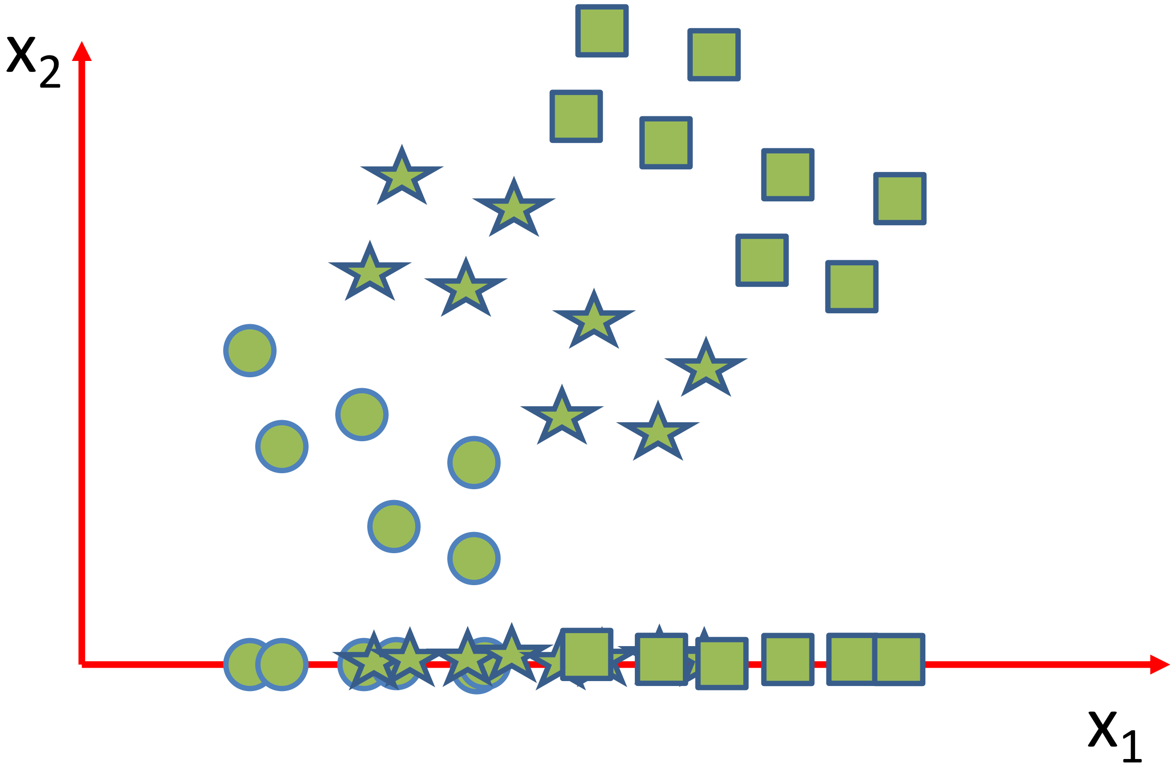
- This method is linear and unsupervised



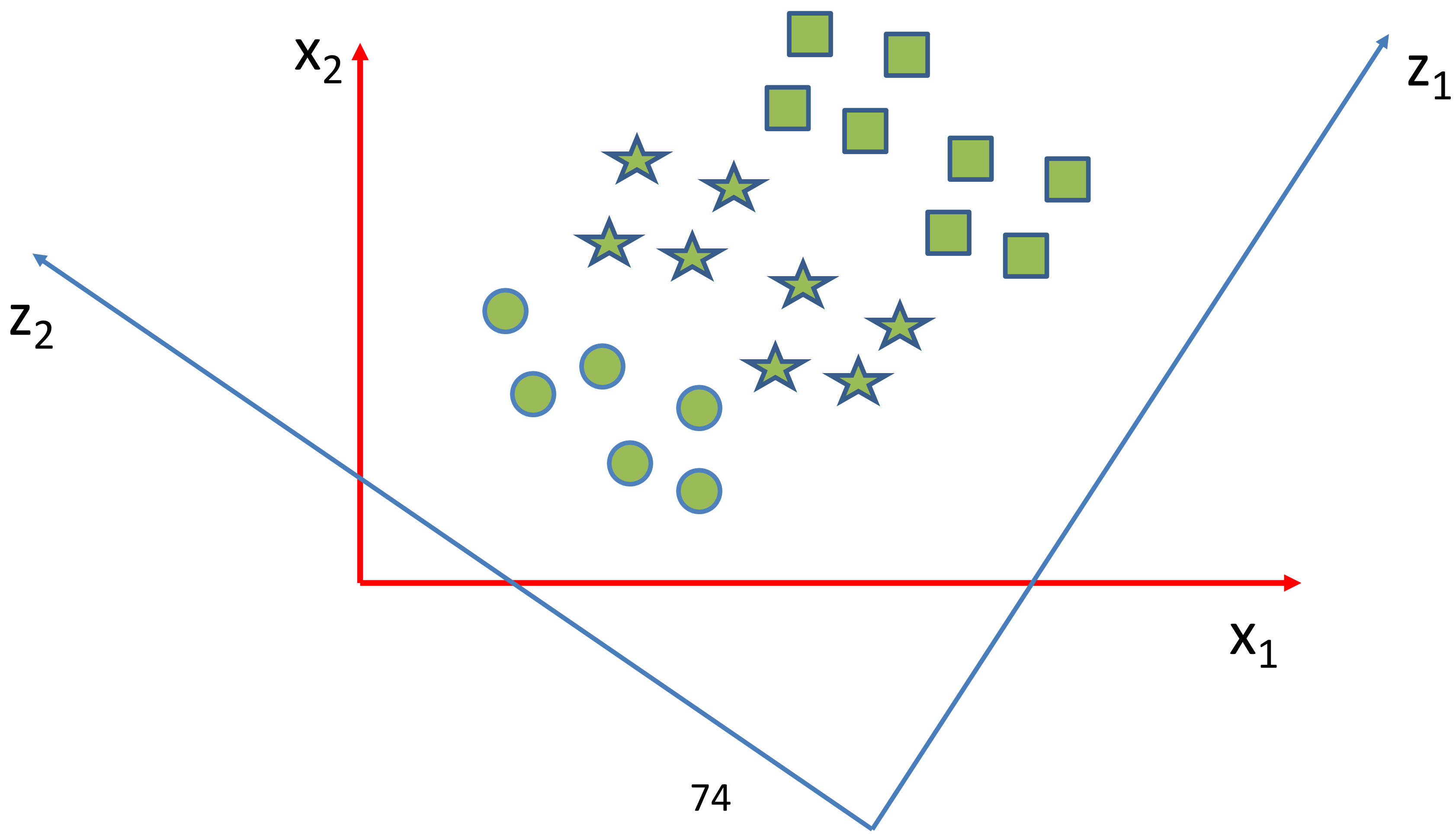
LDA



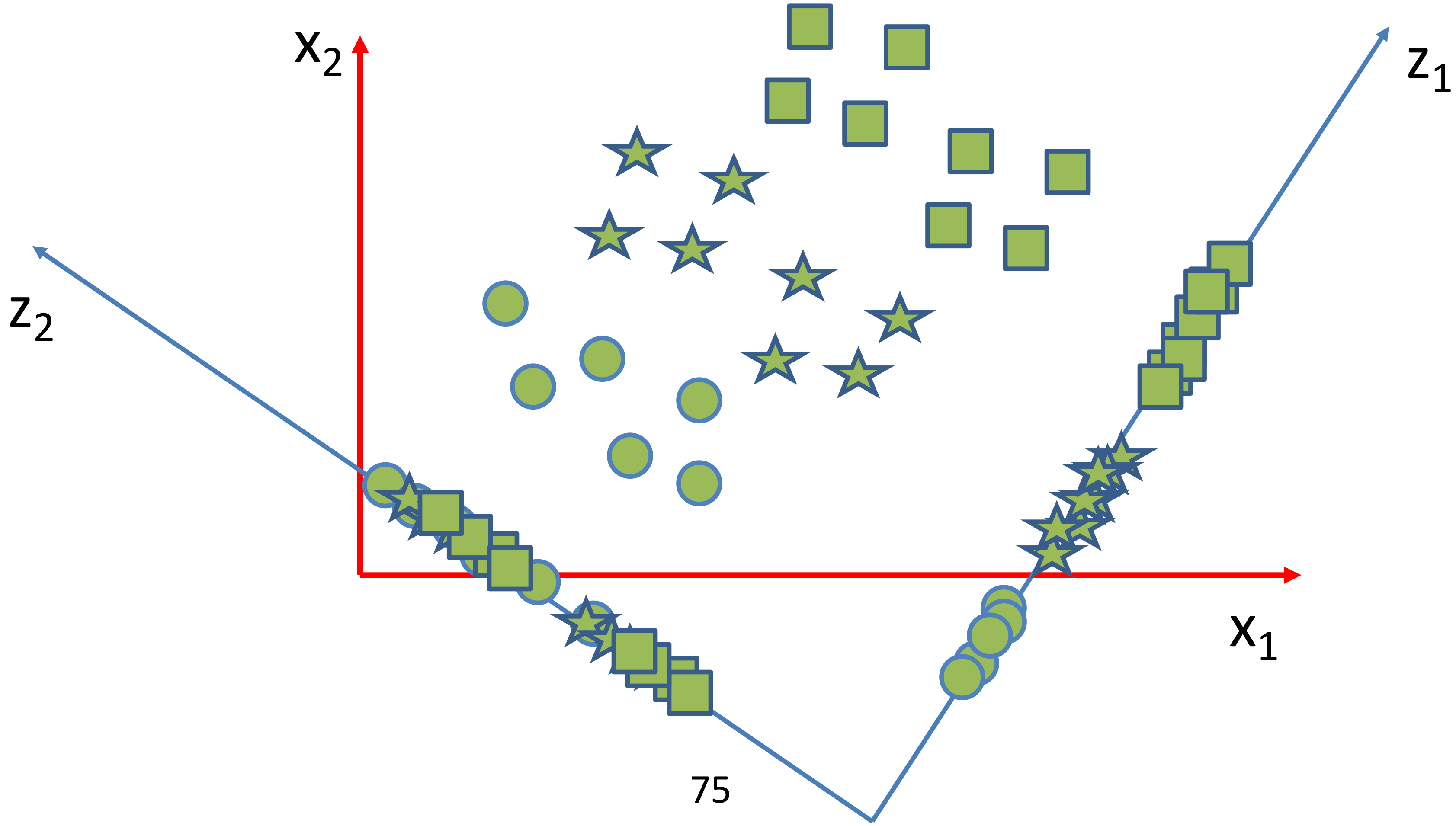
LDA



LDA

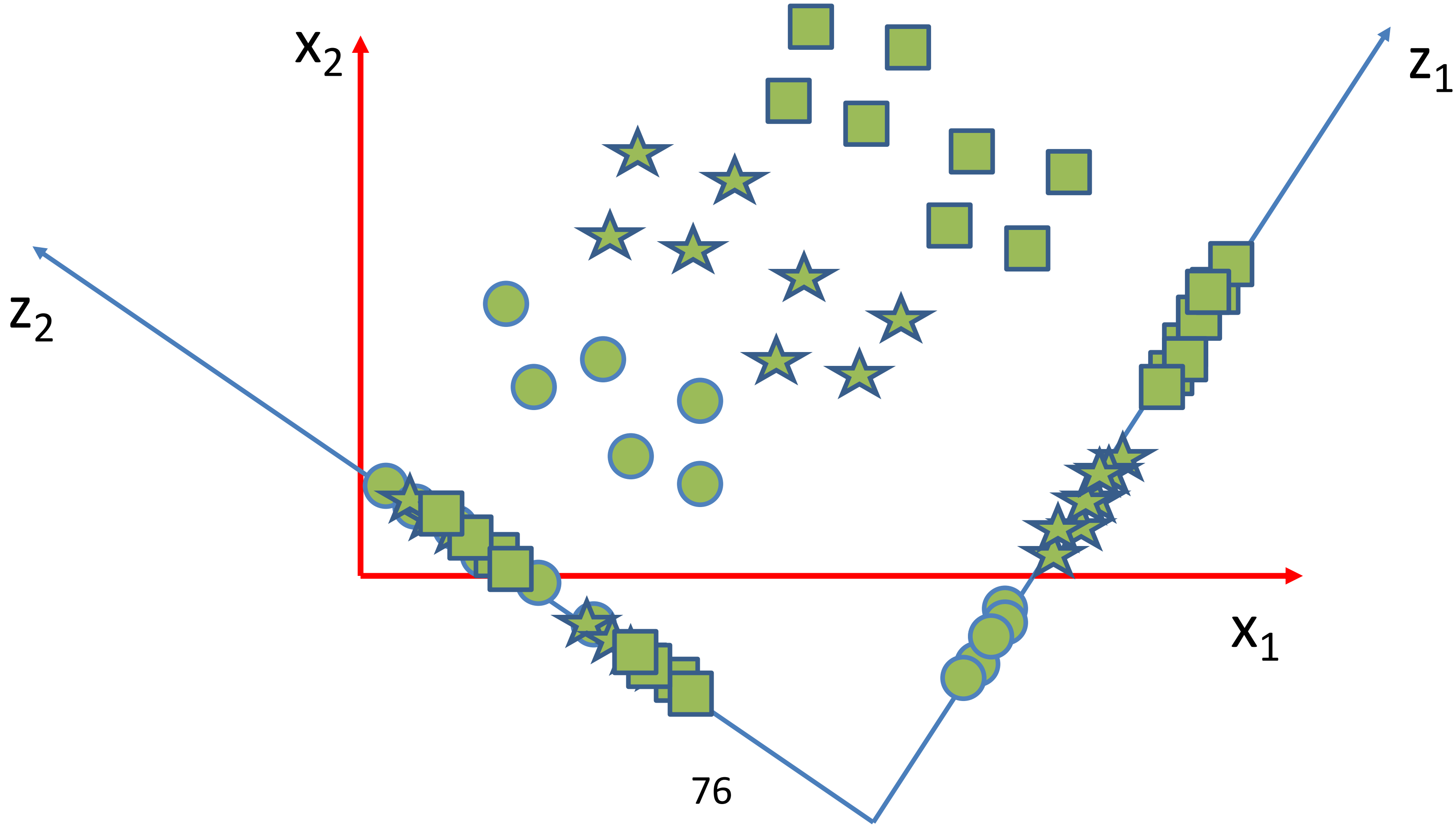


LDA



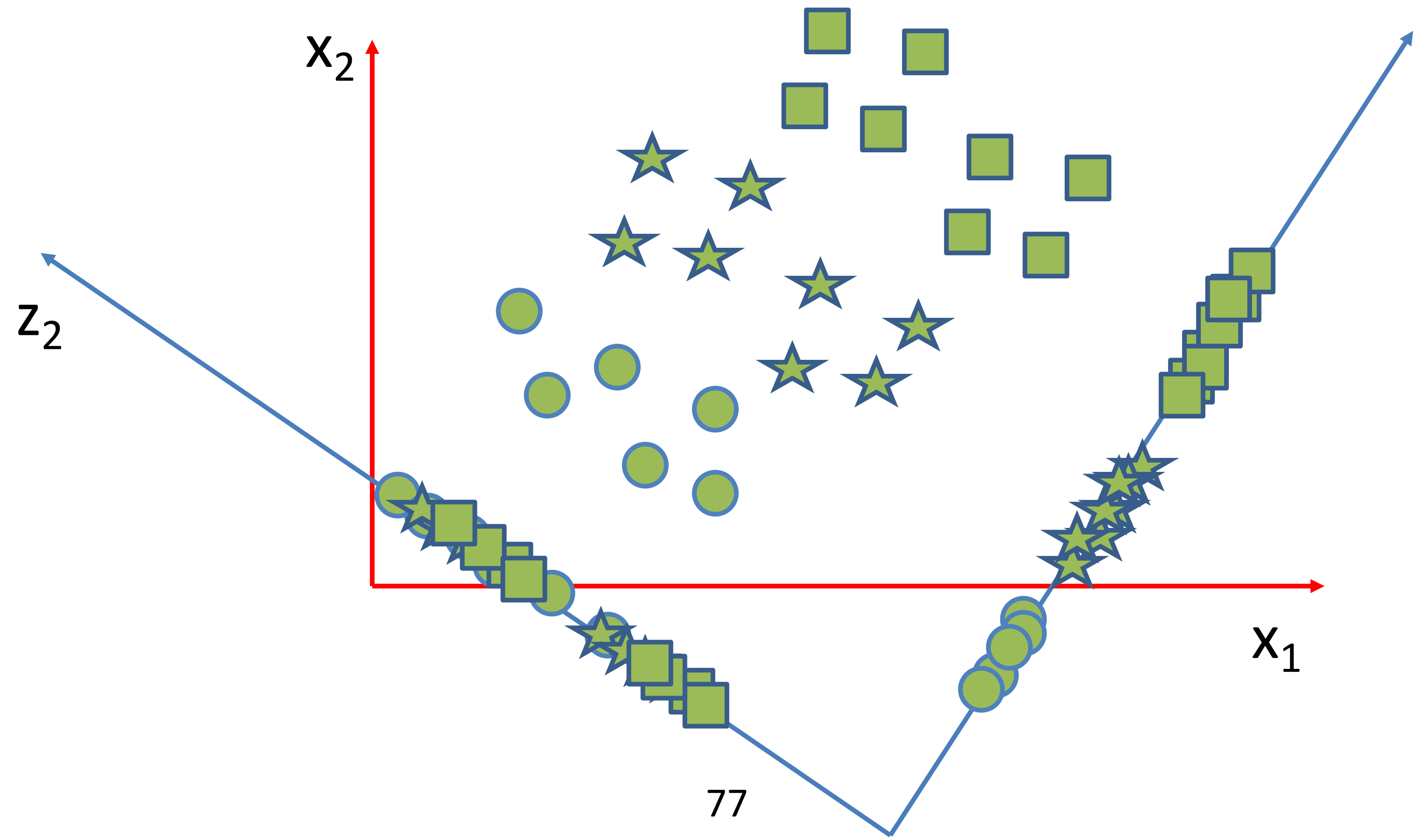
LDA

- Minimize the variance within classes and maximize the distance between classes



LDA

- Linear, supervised



Live Voting

Q&A?

Thanks for listening!