Machine Learning
Theory

Christian Rossi

Academic Year 2023-2024

Abstract

The course topics are:

- Introduction: basic concepts.
- Learning theory:
  - Bias/variance tradeoff. Union and Chernoff/Hoeffding bounds.
  - VC dimension. Worst case (online) learning.
  - Practical advice on how to use learning algorithms.
- Supervised learning:
  - Supervised learning setup. LMS.
  - Logistic regression. Perceptron. Exponential family.
  - Kernel methods: Radial Basis Networks, Gaussian Processes, and Support Vector Machines.
  - Model selection and feature selection.
  - Ensemble methods: Bagging, boosting.
  - Evaluating and debugging learning algorithms.
- Reinforcement learning and control:
  - MDPs. Bellman equations.
  - Value iteration and policy iteration.
  - TD, SARSA, Q-learning.
  - Value function approximation.
  - Policy search. Reinforce. POMDPs.
  - Multi-Armed Bandit.

Contents

CHAPTER 1

Introduction

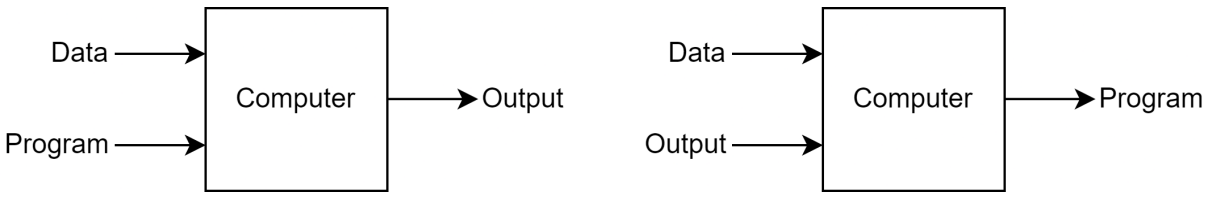
1.1 Machine learning

Definition (*Learning*)**.** A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, improves with experience E.

Machine learning, a subset of artificial intelligence, derives knowledge from experience and induction.

In machine learning, we depend on computers to make informed decisions using new, unfamiliar data. Designing a comprehensive set of meaningful rules can prove to be exceedingly difficult. Machine learning facilitates the automatic extraction of relevant insights from historical data and effectively applies them to new datasets.

The objective is to automate the programming process for computers, acknowledging the bottleneck presented by writing software. Instead, our aim is to utilize the data itself to accomplish the required tasks.



(a) Programming (b) Machine learning

Figure 1.1: Difference between programming and machine learning

Machine learning paradigms can be categorized into three main types:

- *Supervised learning*: involves labeled data and direct feedback, aiming to predict outcomes or future events.
- *Unsupervised learning*: operates without labeled data or feedback, focusing on discovering hidden structures within the data.
- *Reinforcement learning*: centers around a decision-making process, incorporating a reward system to learn sequences of actions.

1.1.1 Supervised learning

Supervised learning encompasses several distinct tasks:

- *Classification*: this involves assigning predefined categories or labels to data points based on their features. The model is trained on labeled data, learning patterns to predict the class labels of new data points.
- *Regression*: the goal here is to predict continuous numerical values based on input features, as opposed to discrete class labels in classification. The model learns a function mapping input features to output values.
- *Probability estimation*: this task predicts the likelihood of certain events or outcomes occurring, often used to gauge the confidence of model predictions.

Formally, in supervised learning, a model learns from data to map known inputs to known outputs. The training set is denoted as $\mathcal{D} = \{\langle x, t\rangle\}$, Where $t = f(x)$, with f representing the unknown function to be determined using supervised learning techniques.

Various techniques can be employed for supervised learning, including linear models, artificial neural networks, support vector machines, and decision trees.

1.1.2 Unsupervised learning

Unsupervised learning encompasses two main tasks:

- *Clustering*: in this task, the objective is to group similar data points together based on their features, without predefined labels. The goal is to uncover underlying patterns or structures within the data. Clustering algorithms segment the data into clusters or groups, where data points within the same cluster exhibit greater similarity compared to those in different clusters. Unlike supervised learning, where labeled data is provided, clustering algorithms explore data solely based on features to identify similarities.
- *Dimensionality reduction*: this task involves reducing the number of input variables or features in a dataset while retaining essential information. This is often done to address the curse of dimensionality, enhance computational efficiency, and mitigate overfitting risks in models. Dimensionality reduction techniques aim to transform high-dimensional data into a lower-dimensional representation while preserving most relevant information.

Formally, in unsupervised learning, computers learn previously unknown patterns and efficient data representations. The training set is defined as $\mathcal{D} = \{x\}$, where the goal is to find a function f that extracts a representation or grouping of the data.

Various techniques are used for unsupervised learning, including k-means clustering, self-organizing maps, and principal component analysis.

1.1.3 Reinforcement learning

Reinforcement learning encompasses several key approaches:

- *Markov decision process*: a mathematical framework for modeling decision-making, involving states, actions, transition probabilities, and rewards. The goal is to find a policy that maximizes cumulative rewards while considering uncertainty.

- *Partially observable MDP*: an extension of MDP where the current state is uncertain and must be inferred from observations. The objective remains the same, but the agent maintains a belief over possible states based on observations.
- *Stochastic games*: models for decision-making with multiple agents, where outcomes depend on actions and random factors. Players aim to optimize strategies considering other players' actions and uncertainties.

In reinforcement learning, the computer learns the optimal policy based on a training set $\mathcal{D}$ containing tuples $\langle x, u, x', r \rangle$, where x is the input, u is the action, x' is the resulting state after the action, and r is the reward. The policy Q^* is defined to maximize $Q^*(x, u)$ over actions u for each state x in the training set.

Various techniques such as Q-learning, SARSA, and fitted Q-iteration are used to find this optimal policy.

CHAPTER 2

Supervised learning

2.1 Introduction

Supervised learning stands as the predominant and well-established learning approach. Its core objective is to enable a computer, given a training set $\mathcal{D} = \{\langle x, t \rangle\}$, to approximate a function f that maps an input x to an output t. The input variables x, often referred to as features or attributes, are paired with output variables t, also known as targets or labels. The tasks undertaken in supervised learning are as follows:

- *Classification*: when t is discrete.
- *Regression*: when t is continuous.
- *Probability estimation*: when$t$ represents a probability.

Supervised learning finds application in scenarios where:

- Humans lack the capability to perform the task directly (e.g., DNA analysis).
- Humans can perform the task but lack the ability to articulate the process (e.g., medical image analysis).
- The task is subject to temporal variations (e.g., stock price prediction).
- The task demands personalization (e.g., movie recommendation).

2.1.1 Function approximation

The process of approximating a function f from a dataset $\mathcal{D}$ involves several steps:

1. *Define a loss function* $\mathcal{L}$: this function calculates the discrepancy between f and h, a chosen approximation.
2. *Select a hypothesis space* $\mathcal{H}$: this space consists of a set of candidate functions from which to choose an approximation h.
3. *Minimize* $\mathcal{L}$ *within* $\mathcal{H}$: the goal is to find an approximation h within the hypothesis space $\mathcal{H}$ that minimizes the loss function $\mathcal{L}$.

The hypothesis space $\mathcal{H}$ can be expanded to theoretically achieve a perfect approximation of the function f. However, a significant challenge arises because the loss function $\mathcal{L}$ cannot be easily determined, primarily due to the absence of the actual function f.

2.1.2 Taxonomy

The taxonomy is as follows:

- *Parametric* or *nonparametric*: parametric methods are characterized by having a fixed and finite number of parameters, while nonparametric methods have a number of parameters that depend on the training set.
- *Frequentist* or *Bayesian*: frequentist approaches utilize probabilities to model the sampling process, whereas Bayesian methods use probability to represent uncertainty about the estimate.
- *Empirical risk minimization* or *structural risk minimization*: empirical risk refers to the error over the training set, while structural risk involves balancing the training error with model complexity.

The type of machine learning can be:

- *Direct*: This method involves learning an approximation of f directly from the dataset $\mathcal{D}$.
- *Generative*: in this approach, the model focuses on modeling the conditional density $\mathrm{P}(t|x)$ and then marginalizing to find the conditional mean:

$$\mathbb{E}\left[t|x\right] = \int t \cdot \mathrm{P}(t|x)dt$$

- *Discriminative*: This method models the joint density $\mathrm{P}(x,t)$, infers the conditional density $\mathrm{P}(t|x)$, and then marginalizes to find the conditional mean:

$$\mathbb{E}\left[t|x\right] = \int t \cdot \mathrm{P}(t|x)dt$$

2.2 Linear regression

The goal of regression is to approximate a function $f(x)$ that maps input x to a continuous output t from a dataset $\mathcal{D}$:

$$\mathcal{D} = \{\langle x, t\rangle\} \implies t = f(x)$$

Here, x is a vector. To perform regression, we assume the existence of a function capable of performing this mapping. The key components of constructing a linear regression problem include:

- The method used to model the function f (the hypothesis space).
- The evaluation criteria for the approximation (the loss function).
- The optimization process for optimizing the model.

In linear regression, the function f is modeled using linear functions. This choice is motivated by several factors:

- Linear models are easily interpretable, making them suitable for explanation.
- Linear regression problems can be solved analytically, allowing for efficient computation.
- Linear functions can be extended to model nonlinear relationships.
- More sophisticated methods often build upon or incorporate elements of linear regression.

Hypothesis space In mathematical terms, the approximation y can be defined as:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{D-1} w_j x_j = \mathbf{w}^T \mathbf{x}$$

Here, $\mathbf{x} = (1, x_1, \ldots, x_{D-1})$ is a vector, and w_0 is called the bias parameter. It's important to note that the output y is a scalar value.

In a two-dimensional space, our hypothesis space will be the set of all points in the plane (w_0, w_1). The coordinates of each point will correspond to a line in the $(\mathbf{x}, y)$ space.

Loss function A commonly used error loss function for the linear regression problem is the sum of squared errors (SSE), defined as:

$$L(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \left(y(x_n, \mathbf{w}) - t_n\right)^2$$

This sum is also referred to as the residual sum of squares (RSS) and can be expressed as the sum of squared residual errors:

$$RSS(\mathbf{w}) = \left\|\boldsymbol{\epsilon}_2^2\right\| = \sum_{i=1}^{N} \epsilon_i^2$$

This formulation of the loss function allows for obtaining a closed-form optimization solution.

Optimization For linear models, a closed-form optimization of the RSS, known as least squares, begins with the matrix representation of the loss function:

$$L(\mathbf{w}) = \frac{1}{2} RSS(\mathbf{w}) = \frac{1}{2} (\mathbf{t} - \boldsymbol{\Phi}_\mathbf{w})^T (\mathbf{t} - \boldsymbol{\Phi}_\mathbf{w})$$

Here, $\boldsymbol{\Phi} = \begin{bmatrix} \phi(x_1) & \cdots & \phi(x_N) \end{bmatrix}^T$ and $\mathbf{t} = \begin{bmatrix} t_1 & \cdots & t_n \end{bmatrix}^T$. To find the optimal $\mathbf{w}$, we compute the first derivative of $L(\mathbf{w})$ and set it to zero:

$$\hat{\mathbf{w}}_{OLS} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T \mathbf{t}$$

However, the inversion of the matrix $\boldsymbol{\Phi}^T \boldsymbol{\Phi}^{-1}$ can be computationally expensive, especially for large datasets, with a complexity of $O(nm^2 + m^3)$, assuming the matrix is non-singular (invertible).

To mitigate this, stochastic gradient descent (SGD) can be employed. The algorithm known as least mean squares (LMS) uses the following update rule:

$$L(\mathbf{x}) = \sum_n L(x_n)$$

Expanding this, we get:

$$\begin{aligned}\mathbf{w}^{(n+1)} &= \mathbf{w}^{(n)} - \alpha^{(n)} \nabla L(x_n) \\ &= \mathbf{w}^{(n)} - \alpha^{(n)} \left(\mathbf{w}^{(n)^T} \phi(\mathbf{x}_n) - t_n \right) \phi(\mathbf{x}_n)\end{aligned}$$

Here, α is the learning rate, and convergence is guaranteed if $\sum_{n=0}^{\infty} = +\infty$ and $\sum_{n=0}^{\infty} = \alpha^{(n)^2} < +\infty$.

If the regression problem involves multiple outputs, meaning that $\mathbf{t}$ is not a scalar, we can solve each regression problem independently. However, we can still use the same set of basis functions. The solution for the weight vectors for all outputs can be expressed as:

$$\hat{\mathbf{W}} = \left(\mathbf{\Phi}^T \mathbf{\Phi} \right)^{-1} \mathbf{\Phi}^T \mathbf{T}$$

Here, each column of matrix $\mathbf{T}$ and $\hat{\mathbf{W}}$ corresponds to the target vector and the weight vector for each output, respectively. This solution can be easily decoupled for each output k:

$$\hat{\mathbf{w}}_k = \left(\mathbf{\Phi}^T \mathbf{\Phi} \right)^{-1} \mathbf{\Phi}^T \mathbf{t}_k$$

An advantage of this approach is that $\left(\mathbf{\Phi}^T \mathbf{\Phi} \right)^{-1}$ only needs to be computed once, regardless of the number of outputs.

2.2.1 Basis function

While a linear combination of input variables may not always suffice to model data, we can still construct a regression model that is linear in its parameters. This can be achieved by defining a model using non-linear basis functions, expressed as:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$$

Here, the components of the vector $\boldsymbol{\phi}(\mathbf{x}) = (1, \phi_1(\mathbf{x}), \ldots, \phi_{M-1}(\mathbf{x}))^T$ are referred to as features. These features allow for a more flexible representation of the input data, enabling the model to capture non-linear relationships between the input variables and the output.

Example:
Let's reconsider a set of data regarding individuals' weight and height, along with their completion times for a one-kilometer run:

Height (cm)	**Weight (kg)**	**Completion time (s)**
180	70	180
184	80	220
174	60	170

We can model this problem using a dummy variable and introduce the Body Mass Index (BMI) as a new feature:

Dummy variable	**Height (cm)**	**Weight (kg)**	**BMI**	**Completion time (s)**
x_0	x_1	x_2	x_3	t
1	180	70	21	180
1	184	80	23	220
1	174	60	20	170

Here, the dummy variable x_0 is always initialized to one. Now, we have the option to retain or discard the weight and height variables, considering only the BMI values for analysis.

The most commonly used basis functions in regression are:

- *Polynomial*:
$$\phi_j(x) = x^j$$
- *Gaussian*:
$$\phi_j(x) = \exp\left(-\frac{(x-\mu_j)^2}{2\sigma^2}\right)$$
- *Sigmoidal*:
$$\phi_j(x) = \frac{1}{1+\exp\left(\frac{\mu_j - x}{\sigma}\right)}$$

Here, the constant μ_j is referred to as a hyperparameter, as its value needs to be determined through experimentation and depends on the user's experience.

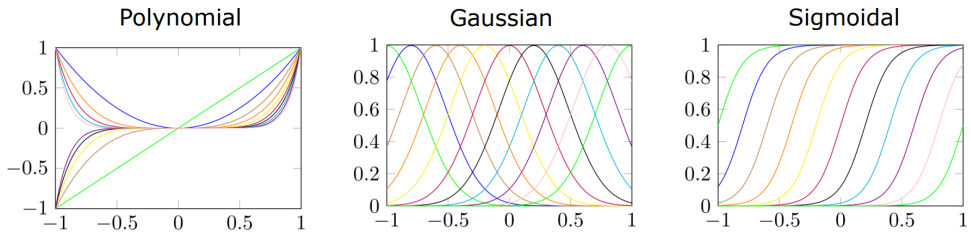


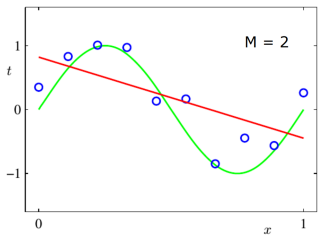
Figure 2.1: Some possible basis functions shapes

It's noteworthy that the Gaussian basis function allows for a local approximation by omitting values that are close to zero. This approach enables capturing the relationship between the input and output in a reduced input space area. As we move away from the mean, approaching zero, the values become negligible.

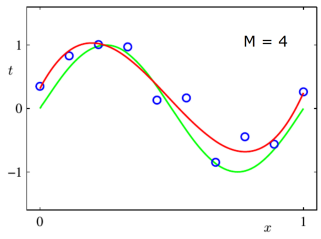
2.2.2 Regularization

A function can achieve a better approximation by increasing the degree of the polynomial used in the regression.

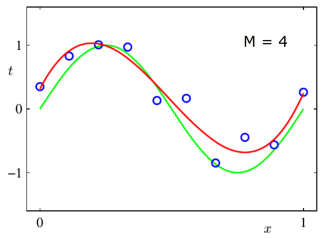
Example:
Consider a function generating a set of points with some noise:



Using a second-order polynomial instead of a linear one provides a better approximation:



Further improving the approximation can be achieved with a higher-degree polynomial (e.g., ninth grade):



However, increasing the polynomial degree also increases the complexity of the model parameters. To address this complexity, adjustments are needed in the loss function:

$$L(\mathbf{w}) = L_D(\mathbf{w}) + \lambda L_W(\mathbf{w})$$

Here, $L_D(\mathbf{w})$ represents the usual loss function, $L_W(\mathbf{w})$ reflects model complexity (a hyperparameter), and λ is the regularization coefficient. $L_W(\mathbf{w})$ can be tailored using ridge regression or lasso methods.

Ridge regression In ridge regression, the regularization term $L_W(\mathbf{w})$ is defined as:

$$L_W(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} = \frac{1}{2}\left\|\mathbf{w}\right\|_2^2$$

Thus, the overall loss function becomes:

$$L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(t_i - \mathbf{w}^T\phi(x_i)\right)^2 + \frac{\lambda}{2}\left\|\mathbf{w}\right\|_2^2$$

Despite the regularization term, the loss function remains quadratic with respect to w, allowing for closed-form optimization:

$$\hat{\mathbf{w}}_{ridge} = \left(\lambda\mathbf{I} + \mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathrm{t}$$

The term $\lambda\mathbf{I}$ is crucial in solving the singularity problem, as it transforms a non-singular matrix into a singular one with an appropriate choice of λ.

Lasso Another common regularization method is lasso, where the regularization term $L_W(\mathbf{w})$ is defined as:

$$L_W(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_1 = \frac{1}{2}\sum_{j=0}^{M-1}|w_j|$$

Thus, the overall loss function becomes:

$$L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(t_i - \mathbf{w}^T\phi(x_i)\right)^2 + \frac{\lambda}{2}\|\mathbf{w}\|_1$$

In this case, closed-form optimization is not possible. However, lasso typically leads to sparse regression models: when the regularization coefficient λ is large enough, some components of $\hat{\mathbf{w}}$ become equal to zero. Regularization can be seen as equivalent to minimizing $L_D(\mathbf{w})$ subject to the constraint:

$$\sum_{j=0}^{M-1}|w_j| \leq \eta$$

2.2.3 Linear regression with probability

We can approach regression in a probabilistic manner by defining a model that probabilistically maps inputs (x) to outputs (t). This model, denoted as $y(x, w)$, incorporates unknown parameters (w). We then model the likelihood, i.e., the probability that observed data $\mathcal{D}$ is generated by a given set of parameters (w), as:

$$\mathrm{P}(\mathcal{D}|\mathbf{w})$$

Finally, we estimate the parameters (w) by maximizing the likelihood:

$$\mathbf{w}_{ML} = \underset{\mathbf{w}}{\operatorname{argmax}}\,\mathrm{P}(\mathcal{D}|\mathbf{w})$$

For linear regression, we define the model as:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon = \mathbf{w}^T\mathbf{\Phi}(\mathbf{x}) + \epsilon$$

Here, we assume a linear model for $y(\mathbf{x}, \mathbf{w})$ and introduce noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Consequently, given a dataset $\mathcal{D}$ of N samples with inputs $\mathbf{X} = \begin{bmatrix}\mathbf{x}_1 & \cdots & \mathbf{x}_n\end{bmatrix}$ and outputs $\mathbf{t} = \begin{bmatrix}t_1 & \cdots & t_n\end{bmatrix}^T$, we have:

$$\mathrm{P}(\mathcal{D}|\mathbf{w}) = \mathrm{P}(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) = \prod_{n=1}^{N}\mathcal{N}(t_n|\mathbf{w}^T\mathbf{\Phi}(\mathbf{x}_n), \sigma^2)$$

To find $\mathbf{w}_{ML}$, it is convenient to maximize the log-likelihood, obtaining:

$$\ell(\mathbf{w}) = \ln\mathrm{P}(t_n|\mathbf{x}_n, \mathbf{w}, \sigma^2) = -\frac{N}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}RSS(\mathbf{w})$$

Notice that the first part of the final formula is a constant independent of $\mathbf{w}$, so it can be ignored in maximizing the likelihood. Solving the optimization problem by setting the gradient to zero $\ell(\mathbf{w}) = 0$, yields the final formula:

$$\mathbf{w}_{ML} = \left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{t}$$

This result aligns with the ordinary least squares approach.

This outcome allows us to interpret ordinary least squares from a probabilistic perspective, confirming that we are utilizing a normally distributed probabilistic function to generate the residuals in OLS.

Bayesian linear regression Bayesian linear regression follows a structured approach:

1. Formulation of probabilistic knowledge:
   (a) Qualitatively define the model expressing our knowledge.
   (b) Incorporate unknown parameters into the model.
   (c) Represent assumptions about these parameters with a prior distribution before observing any data.
2. Data observation.
3. Computation of posterior probability distribution for parameters:
$$\mathrm{P}(parameters|data) = \frac{\mathrm{P}(data|parameters)\mathrm{P}(parameters)}{\mathrm{P}(data)}$$
4. Utilization of Posterior Distribution to:
   - Make predictions by averaging over the posterior distribution.
   - Assess or accommodate uncertainty in parameter values.
   - Make decisions by minimizing expected posterior loss.

The posterior distribution for model parameters is derived by combining the prior with the likelihood for parameters given the data:

$$\mathrm{P}(\mathbf{w}|\mathcal{D}) = \frac{\mathrm{P}(\mathcal{D}|\mathbf{w})\mathrm{P}(\mathbf{w})}{\mathrm{P}(\mathcal{D})}$$

Here, $\mathrm{P}(\mathbf{w})$ represents the prior probability over parameters, $\mathrm{P}(\mathcal{D}|\mathbf{w})$ denotes the likelihood, and $\mathrm{P}(\mathcal{D})$ is the marginal likelihood acting as a normalization constant:

$$\mathrm{P}(\mathcal{D}) = \int \mathrm{P}(\mathcal{D}|\mathbf{w})\mathrm{P}(\mathbf{w})d\mathbf{w}$$

The mode of the posterior, known as the Maximum A Posteriori (MAP) estimate, yields the most probable value of $\mathbf{w}$ given the data.

A Gaussian likelihood assumption allows the prior to be modeled conveniently as a conjugate prior:

$$\mathrm{P}(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_0, \mathbf{S}_0)$$

Consequently, the posterior remains Gaussian:

$$\mathrm{P}(\mathbf{w}|\mathbf{t}, \mathbf{\Phi}, \sigma^2) \propto \mathcal{N}(\mathbf{w}|\mathbf{w}_0, \mathbf{S}_0)\mathcal{N}(\mathbf{t}|\mathbf{\Phi}\mathbf{w}, \sigma^2\mathbf{I})$$

Resulting in:

$$\begin{cases} \mathrm{P}(\mathbf{w}|\mathbf{t}, \mathbf{\Phi}, \sigma^2) = \mathcal{N}(\mathbf{w}|\mathbf{w}_N, \mathbf{S}_N) \\ \mathbf{w}_N = \mathbf{S}_N \left(\mathbf{S}_0^{-1}\mathbf{w}_0 + \dfrac{\mathbf{\Phi}^T\mathbf{t}}{\sigma^2} \right) \\ \mathbf{S}_N^{-1} = \mathbf{S}_0^{-1} + \dfrac{\mathbf{\Phi}^T\mathbf{\Phi}}{\sigma^2} \end{cases}$$

Prior infinitely broad When the prior distribution is infinitely broad, the Maximum A Posteriori (MAP) estimate coincides with the Maximum Likelihood (ML) solution:

$$\begin{cases} \lim_{\mathbf{S}_0 \to \infty} \mathbf{w}_N = \left(\mathbf{\Phi}^T \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^T \mathbf{t} \\ \lim_{\mathbf{S}_0 \to \infty} \mathbf{S}_N^{-1} = \dfrac{\mathbf{\Phi}^T \mathbf{\Phi}}{\sigma^2} \end{cases}$$

This yields the ordinary least squares formula. However, in this case, we also have the covariance matrix, providing information about the related uncertainty. The only missing parameter is σ^2, which can be computed as:

$$\sigma^2 = \frac{1}{N-M} \sum_{n=1}^{N} \left(t_n - \hat{\mathbf{w}}^T (\phi)(\mathbf{x}_n)\right)^2$$

The ML estimate $\mathbf{w}_{ML}$ of $\mathbf{w}$ has the smallest variance among linear unbiased estimates and the lowest Mean Squared Error (MSE) among linear unbiased estimates (Gauss-Markov).

Prior not infinitely broad When the prior distribution is not infinitely broad, such that $\mathbf{w}_0 = 0$ and $\mathbf{S}_0 = \tau^2 \mathbf{I}$, we can express the logarithm of the posterior distribution $\mathrm{P}(\mathbf{w}|\mathbf{t})$ as:

$$\ln \mathrm{P}(\mathbf{w}|\mathbf{t}) = -\frac{1}{2\sigma^2} \sum_{i=1}^{N} \left(t_i - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i)\right)^2 - \frac{1}{2\tau^2} \|\mathbf{w}\|_2^2$$

In this scenario, the Maximum A Posteriori estimate, MAP ($\mathbf{w}N$), coincides with the solution of ridge regression $\hat{\mathbf{w}}ridge$ with a regularization parameter λ set to $\lambda = \frac{\sigma^2}{\tau^2}$.

Sequential learning How to leverage the Bayesian approach for sequential learning:

1. Begin by computing the posterior with the initial data.
2. As additional data becomes available, update the prior with this new information to obtain the updated posterior.

Predictive distribution In a Bayesian framework, one can determine the probability distribution of the target variable for a new sample $\mathbf{x}^*$ (given the training data $\mathcal{D}$) by integrating over the posterior distribution:

$$\mathrm{P}(t^*|\mathbf{x}^*, \mathcal{D}) = \mathbb{E}\left[t^*|\mathbf{x}^*, \mathbf{w}, \mathcal{D}\right] = \int \mathrm{P}(t^*|\mathbf{x}^*, \mathbf{w}, \mathcal{D}) \mathrm{P}(\mathbf{w}|\mathcal{D}) d\mathbf{w}$$

This is commonly referred to as the predictive distribution. However, computing this predictive distribution typically involves the intractable task of determining the posterior distribution. Nevertheless, under certain assumptions, it is possible to compute the predictive distribution as follows:

$$\sigma_N^2(\mathbf{x}) = \sigma^2 + \boldsymbol{\phi}(\mathbf{x})^T \mathbf{S}_N \boldsymbol{\phi}(\mathbf{x})$$

Here, as the number of data points N approaches infinity, the uncertainty associated with the parameters (second term) diminishes, and the variance of the predictive distribution depends solely on the variance of the data (σ^2).

2.2.4 Challenges and limitations

Modeling presents challenges in ensuring our model effectively represents a wide range of plausible functions while maintaining informative priors without overly spreading out probabilities or assigning negligible values.

On the computational side, limitations arise with analytical integration, particularly in cases involving non-conjugate priors and complex models. Approaches like Gaussian (Laplace) approximation, Monte Carlo integration, and variational approximation become necessary for addressing these complexities and achieving accurate results.

Linear models with fixed basis functions offer several benefits:

- They permit closed-form solutions, facilitating efficient computation.
- They lend themselves to tractable Bayesian treatment, enabling principled uncertainty quantification.
- They can capture non-linear relationships by employing appropriate basis functions.

However, these models also come with several drawbacks:

- Basis functions remain static and non-adaptive to variations in the training data.
- These models are susceptible to the curse of dimensionality, particularly when dealing with high-dimensional feature spaces.

2.3 Classification

Linear classification involves learning an approximation of a function $f(x)$ that maps inputs x to discrete classes C_k (with $k = 1, \ldots, K$) from a dataset $\mathcal{D}$:

$$\mathcal{D} = \{\langle x, C_k \rangle\} \implies C_k = f(x)$$

Various approaches to classification include:

- *Discriminant function*: modeling a parametric function that directly maps inputs to classes and learning the parameters from the data.
- *Probabilistic discriminative approach*: designing a parametric model of $\mathrm{P}(C_k|\mathbf{x})$ and learning the model parameters from the data.
- *Probabilistic generative approach*: modeling $\mathrm{P}(\mathbf{x}|C_k)$ and class priors $\mathrm{P}(C_k)$, fitting models to the data, and inferring the posterior using Bayes' rule:

$$\mathrm{P}(C_k|\mathbf{x}) = \frac{\mathrm{P}(\mathbf{x}|C_k)\mathrm{P}(C_k)}{\mathrm{P}(\mathbf{x})}$$

2.3.1 Discriminant function

We begin by examining the linear discriminant functions defined as:

$$f(\mathbf{x}, \mathbf{w}) = f\left(w_0 + \sum_{j=1}^{D-1} w_j x_j\right) = f(\mathbf{x}^T\mathbf{w} + w_0)$$

Here, the function $f(\cdot)$ is not linear in $\mathbf{w}$ due to the presence of the (non-linear) activation function f, which yields either a discrete label or a probability value as its output.

The function $f(\cdot)$ partitions the input space into decision regions, with their boundaries known as decision boundaries or decision surfaces. Notably, these decision surfaces are linear functions of $\mathbf{x}$ and $\mathbf{w}$, expressed as:

$$\mathbf{x}^T\mathbf{w} + w_0 = \text{constant}$$

It's important to note that generalized linear models are more complex to utilize compared to linear models due to the incorporation of non-linear activation functions.

Labels A common encoding for two-class problems involves binary encoding, where $t \in \{0, 1\}$. In this setup, $t = 1$ indicates the positive class, while $t = 0$ denotes the negative one. Using this encoding, both t and $f(\cdot)$ represent the probability of the positive class.

Another encoding option for two-class problems is $t \in \{-1, 1\}$, which is preferable for certain algorithms.

For problems with K classes, a typical choice is the 1-of-K encoding. Here, t is a vector of length K, with a 1 in the position corresponding to the encoded class. In this encoding scheme, both t and $f(\cdot)$ represent the probability density over the classes.

Example:
For instance, in a problem with $K = 5$ classes, a data sample belonging to class 4 would be encoded as:

$$t = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix}^T$$

Two-class problem The most general formulation for a discriminant linear function in a two-class linear problem is:

$$f(\mathbf{x}, \mathbf{w}) = \begin{cases} C_1 & \text{if } \mathbf{x}^T\mathbf{w} + w_0 \geq 0 \\ C_2 & \text{otherwise} \end{cases}$$

From this formulation, we can deduce the following properties:

- The decision boundary is $y(\cdot) = \mathbf{x}^T\mathbf{w} + w_0 = 0$.
- The decision boundary is orthogonal to $\mathbf{w}$.
- The distance of the decision boundary from the origin is $\frac{w_0}{\|\mathbf{w}\|_2}$.
- The distance of the decision boundary from $\mathbf{x}$ is $\frac{y(\mathbf{x})}{\|\mathbf{w}\|_2}$.

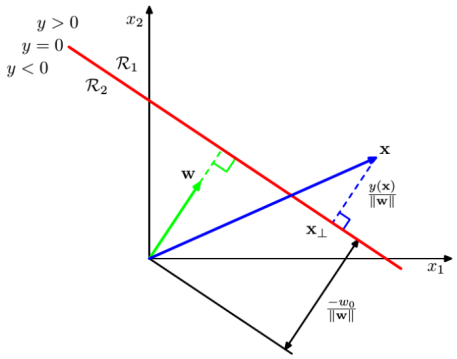


Figure 2.2: Two-class decision problem boundaries

Multiple-class problem In multiple class problems with K classes, various encoding methods can be employed:

- *One versus the rest*: this approach involves using $K-1$ binary classifiers, where each classifier distinguishes between one class (C_i) and the rest of the classes. However, this method introduces ambiguity since there may be regions mapped to multiple classes.
- *One versus one*: this method utilizes $\frac{K(K-1)}{2}$ binary classifiers, where each classifier discriminates between pairs of classes C_i and C_j. Similar to the one versus the rest approach, this method also suffers from ambiguity.

One potential solution to mitigate the ambiguity in multi-class classification is to employ K linear discriminant functions:

$$y_k(\mathbf{x}) = \mathbf{x}^T \mathbf{w}_k + w_{k0} \qquad k = 1, \ldots, K$$

In this approach, an input vector $\mathbf{x}$ is assigned to class C_k if $y_k > y_j$ for all $j \neq k$. This method ensures that the decision boundaries are singly connected and convex.

Linear basis function models Up to this point, we have focused on models operating within the input space. However, we can enhance these models by incorporating a fixed set of basis functions $\boldsymbol{\phi}(\mathbf{x})$. Essentially, this involves applying a non-linear transformation to map the input space into a feature space. Consequently, decision boundaries that are linear within the feature space would correspond to nonlinear boundaries within the input space. This extension enables the application of linear classification models to problems where samples are not linearly separable.

Ordinary least squares Let's consider a K-class problem using a 1-of-K encoding for the target. Each class is modeled with a linear function:

$$y_k(\mathbf{x}) = \mathbf{x}^T \mathbf{w}_k + w_{k0} \qquad k = 1, \ldots, K$$

In matrix notation, this can be expressed as:

$$\mathbf{y}(\mathbf{x}) = \tilde{\mathbf{W}}^T \tilde{\mathbf{x}}$$

Here, $\tilde{\mathbf{W}}$ is of size $(D+1) \times K$, where its k-th column is denoted by $\tilde{\mathbf{w}}_k = \left(w_{k0}, \mathbf{w}_k^T\right)^T$, and $\tilde{\mathbf{x}} = \left(1, \mathbf{x}^T\right)^T$.

Given a dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{t}_i\}$ where $i = 1, \ldots, N$, we can utilize the least squares method to determine the optimal value of $\tilde{\mathbf{W}}$, resulting in:

$$\tilde{\mathbf{W}} = \left(\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}\right) \tilde{\mathbf{X}}^T \tilde{\mathbf{T}}$$

Here, $\tilde{\mathbf{X}}$ is an $N \times (D+1)$ matrix with its i-th row being $\tilde{\mathbf{x}}_i^T$ and $\mathbf{T}$ is an $N \times K$ matrix with its i-th row as $\mathbf{t}_i^T$. In this setup, any new sample $\tilde{\mathbf{x}}_{new}^T$ is assigned to class C_k if $t_k > t_j$ for all j, where t_k represents the k-th component of the model output computed as $t_k = \tilde{\mathbf{x}}^T \tilde{\mathbf{w}}_k$.

The primary challenge with employing ordinary least squares in classification is that the resulting decision boundaries between regions can vary significantly based on the distribution of the data. This method may yield effective or suboptimal boundaries depending on the characteristics of the dataset.

Perceptron To address the issue of poor boundaries, one approach is to utilize a model known as the perceptron. Proposed by Rosenblatt in 1958, the perceptron is a linear discriminant model designed specifically for two-class problems, with class encoding as $\{-1, 1\}$. The perceptron model is defined as:

$$f(\mathbf{x}, \mathbf{w}) = \begin{cases} +1 & \text{if } \mathbf{x}^T\mathbf{w} + w_0 \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

The perceptron algorithm aims to determine a decision surface, also known as a separating hyperplane, by minimizing the distance of misclassified samples to the boundary. This minimization of the loss function can be achieved using stochastic gradient descent.

Although simpler loss functions could theoretically be used, they are often more complex to minimize in practice. Therefore, stochastic gradient descent is commonly employed for optimization in perceptron learning.

The core concept of the perceptron is to optimize $\mathbf{w}$ such that $\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i) \geq 0$ for $\mathbf{x}_i \in C_1$ and $\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i) < 0$ otherwise. The perceptron criterion is expressed as:

$$L_P\mathbf{w} = -\sum_{n\in\mathcal{M}} \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n)t_n$$

Here, correctly classified samples do not contribute to L, and each misclassified sample $\mathbf{x}_i \in \mathcal{M}$ contributes as $\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n)t_n$.

Minimizing L_P is achieved using stochastic gradient descent:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha\nabla L_P(\mathbf{w}) = \mathbf{w}^{(k)} + \alpha\boldsymbol{\phi}(\mathbf{x}_n)t_n$$

Since the scale of $\mathbf{w}$ does not affect the perceptron function, the learning rate α is often set to 1. The perceptron algorithm takes a dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{t}_i\}$ where $i = 1, \ldots, N$.

Algorithm 1 Perceptron algorithm

```
Initialize w_0
k ← 0
repeat
    k ← k + 1
    n ← k mod N
    if then t̂_n ≠ t_n
        w_{k+1} ← w_k + φ(X_n)t_n
    end if
until convergence
```

Theorem 2.3.1 (Perceptron convergence)**.** *If the training dataset is linearly separable in the feature space* $\boldsymbol{\Phi}$*, then the perceptron learning algorithm is guaranteed to find an exact solution in a finite number of steps.*

Several steps may be necessary, making it challenging to distinguish between non-separable problems and slowly converging ones. If multiple solutions exist, the one obtained by the algorithm depends on the parameter initialization and the order of updates.

2.3.2 Probabilistic discriminative approaches

In a discriminative approach, we model the conditioned class probability directly:

$$\mathrm{P}(C_1|\boldsymbol{\phi}) = \frac{1}{1 + e^{-\mathbf{w}^T\boldsymbol{\phi}}} = \sigma(\mathbf{w}^T\boldsymbol{\phi})$$

Here, $\sigma(\cdot)$ denotes the sigmoidal function. This model is commonly referred to as logistic regression.

Maximum likelihood Given a dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$, where $i = 1, \ldots, N$ and $t_i \in \{0, 1\}$, we aim to maximize the likelihood, i.e., the probability of observing the targets given the inputs $\mathrm{P}(\mathbf{t}|\mathbf{X}, \mathbf{w})$. We model the likelihood of a single sample using a Bernoulli distribution, employing the logistic regression model for conditioned class probability:

$$\mathrm{P}(t_n|\mathbf{x}_n, \mathbf{w}) = y_n^{t_n}(1 - y_n)^{1-t_n} \qquad y_n = \mathrm{P}(t_n = 1|\mathbf{x}_n, \mathbf{w}) = \sigma(\mathbf{w}^T\boldsymbol{\phi}_n)$$

Assuming independent sampling of data in $\mathcal{D}$, we have:

$$\mathrm{P}(\mathbf{t}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n}(1 - y_n)^{(1-t_n)} \qquad y_n = \sigma(\mathbf{w}^T\boldsymbol{\phi}_n)$$

The negative log-likelihood (also known as cross-entropy error function) serves as a convenient loss function to minimize:

$$L(\mathbf{w}) = -\ln \mathrm{P}(\mathbf{t}|\mathbf{X}, \mathbf{w}) = -\sum_{n=1}^{N} (t_n \ln y_n + (1 - t_n) \ln(1 - y_n)) = \sum_{n=1}^{N} L_n$$

The derivative of L yields the gradient of the loss function:

$$\nabla L(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\, \boldsymbol{\phi}_n$$

Due to the nonlinearity of the logistic regression function, a closed-form solution is not feasible. Nevertheless, the error function is convex, allowing for gradient-based optimization (even in an online learning setting).

Multi class logistic regression In multi class problems, $\mathrm{P}(C_k|\boldsymbol{\phi})$ is modeled by applying a softmax transformation to the output of K linear functions (one for each class):

$$\mathrm{P}(C_k|\boldsymbol{\phi}) = y_k(\boldsymbol{\phi}) = \frac{e^{\mathbf{w}_k^T\boldsymbol{\phi}}}{\sum_j e^{\mathbf{w}_j^T\boldsymbol{\phi}}}$$

Similar to the two-class logistic regression and assuming 1-of-K encoding for the target, we compute the likelihood as:

$$\mathrm{P}(\mathbf{T}|\boldsymbol{\Phi}, \mathbf{w}_1, \ldots, \mathbf{w}_K) = \prod_{n=1}^{N} \left(\prod_{k=1}^{K} \mathrm{P}(C_k|\boldsymbol{\phi}_n)^{t_{nk}} \right) = \prod_{n=1}^{N} \left(\prod_{k=1}^{K} y_{nk}^{t_{nk}} \right)$$

As in the two-class problem, we minimize the cross-entropy error function:

$$L(\mathbf{w}_1, \ldots, \mathbf{w}_K) = -\ln \mathrm{P}(\mathbf{T}|\mathbf{\Phi}, \mathbf{w}_1, \ldots, \mathbf{w}_K) = -\sum_{n=1}^{N} \left(\sum_{k=1}^{K} t_{nk} \ln y_{nk} \right)$$

Then, we compute the gradient for each weight vector:

$$\nabla L_{\mathbf{w}_j}(\mathbf{w}_1, \ldots, \mathbf{w}_K) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\, \boldsymbol{\phi}_n$$

Perceptron Replacing the logistic function with a step function in logistic regression yields the same updating rule as the perceptron algorithm.

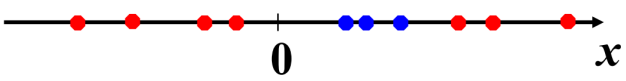
2.4 Kernel methods

Frequently, we seek to detect nonlinear patterns within our datasets. In nonlinear regression, the connection between input and output may deviate from linearity, while in nonlinear classification, class boundaries might not be linearly separable. Linear models often prove insufficient in capturing such complexities. However, kernel methods offer a solution by transforming data into higher-dimensional spaces where linear relationships become apparent, thereby enabling linear models to effectively operate in nonlinear scenarios.

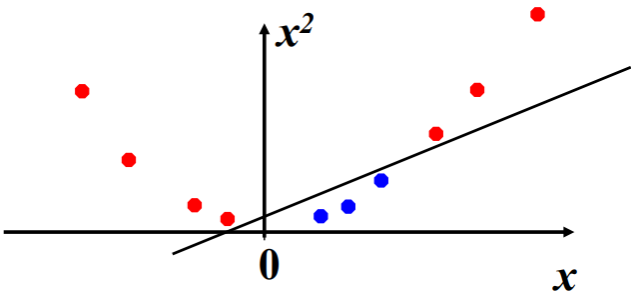
The process of transforming the original input space into a feature space is termed feature mapping, denoted as:

$$\Phi : x \to \phi(x)$$

Example:
Consider a binary classification problem where no linear separator exists:



Now, let's map the input space (a single variable x) to a feature space with two features: $x \to \{x, x^2\}$ As a result, the data becomes linearly separable:



This concept extends naturally to higher dimensions and more intricate problem domains.

However, a significant drawback arises known as the curse of dimensionality. This occurs due to the exponential growth in the number of features as the input variables increase, rendering the mapping computationally infeasible. Kernel methods offer a solution to this challenge

by bypassing the need for explicit computation of the feature mapping. While they are computationally intensive, they remain feasible for practical implementation.

2.4.1 Kernel function

The kernel function is defined as the scalar product between the feature vectors of two data samples:

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x})$$

The kernel function exhibits symmetry: $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$. It can be interpreted as a measure of similarity between $\mathbf{x}$ and $\mathbf{x}'$.

Interestingly, very large feature vectors, even infinite ones, can result in a kernel function that is computationally tractable.

Certain special classes of kernel functions exist:

- *Stationary kernels*: $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}')$.
- *Homogeneous kernels* (or radial basis functions): $k(\mathbf{x}, \mathbf{x}') = k(\|\mathbf{x} - \mathbf{x}'\|)$.

Kernel function design We are not obligated to compute the kernel function by first generating the feature space, as we aim to avoid explicitly calculating the feature vectors. Two primary approaches exist for designing a kernel function:

- Create kernel functions directly from scratch.
- Design kernel functions by applying a predefined set of rules to existing ones.

In both cases, it's crucial to ensure that the resulting kernel functions are valid, meaning they correspond to a scalar product in some feature space.

Theorem 2.4.1 (Mercer)**.** *Any continuous, symmetric, positive semi-definite kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$ can be expressed as a dot product in a high-dimensional space.*

For this theorem, the necessary and sufficient condition for a function $k(\mathbf{x}, \mathbf{x}')$ to be a valid kernel is that the Gram matrix $\mathbf{K}$ is positive semi-definite for all possible choices of $\mathcal{D} = \{x_i\}$. This condition implies that $\mathbf{x}^T \mathbf{K} \mathbf{x} > 0$ for any non-zero real vector $\mathbf{x}$, meaning that the double sum $\sum_i \sum_j \mathbf{K}_{ij} \mathbf{x}_i \mathbf{x}_j$ is strictly positive for any real numbers $\mathbf{x}_i$ and $\mathbf{x}_j$.

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$ the following rules can be applied to design a new valid kernel:

1. $k_(\mathbf{x}, \mathbf{x}') = c k_1(\mathbf{x}, \mathbf{x}')$, where $c > 0$ is a constant.
2. $k_(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}')$, where $f(\cdot)$ is any function.
3. $k_(\mathbf{x}, \mathbf{x}') = q\left(k_1(\mathbf{x}, \mathbf{x}')\right)$, where $q(\cdot)$ is a polynomial with non-negative coefficients.
4. $k_(\mathbf{x}, \mathbf{x}') = e^{k_1(\mathbf{x}, \mathbf{x}')}$.
5. $k_(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$.
6. $k_(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}')$.
7. $k_(\mathbf{x}, \mathbf{x}') = k_3(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}'))$, where $\boldsymbol{\phi}(\mathbf{x})$ maps $\mathbf{x}$ to $\mathbb{R}^M$ and $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$.

8. $k_(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{A} \mathbf{x}$, where $\mathbf{A}$ is a symmetric semidefinite matrix.
9. $k_(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b)$.
10. $k_(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a) k_b(\mathbf{x}_b, \mathbf{x}'_b)$.

Kernel trick It's feasible to modify the representation of linear models by substituting terms involving $\phi(\mathbf{x})$ with alternatives solely based on $k(\mathbf{x}, \cdot)$. In essence, the output of a linear model can be computed solely based on the similarities between data samples, as computed with the kernel function.

This methodology, known as the kernel trick, finds application in various learning algorithms including: ridge regression, $K-NN$ regression, perceptron, nonlinear PCA, and support vector machines.

Gaussian kernel The Gaussian kernel is a widely employed kernel function in various machine learning algorithms. Its mathematical representation is given by:

$$k(\mathbf{x}, \mathbf{x}') = e^{-\frac{\|\mathbf{x}-\mathbf{x}'\|}{2\sigma^2}}$$

This kernel function defines a similarity measure between two vectors $\mathbf{x}$ and $\mathbf{x}'$ in the feature space. It assigns higher similarity to vectors that are closer to each other, based on the Euclidean distance, with σ controlling the width of the kernel.

Additionally, the Gaussian kernel can be generalized by replacing the dot product $\mathbf{x}^T\mathbf{x}'$ with a nonlinear kernel function $\kappa(\mathbf{x}, \mathbf{x}')$. This leads to the extended form of the Gaussian kernel:

$$k(\mathbf{x}, \mathbf{x}') = e^{-\frac{\kappa(\mathbf{x},\mathbf{x}) + \kappa(\mathbf{x}',\mathbf{x}') - 2\kappa(\mathbf{x},\mathbf{x}')}{2\sigma^2}}$$

This extension allows the Gaussian kernel to operate in a more flexible feature space, potentially capturing nonlinear relationships between data points, thereby enhancing its applicability in various machine learning tasks.

Symbolic data kernel Kernel methods are not limited to real vectors as inputs; they can be extended to various data structures such as graphs, sets, strings, texts, and more. The kernel function serves as a measure of similarity between two samples. For example, in the case of sets, a common kernel function is employed:

$$k(A_1, A_2) = 2^{|A_1 \cap A_2|}$$

This kernel function quantifies the similarity between two sets A_1 and A_2 by computing the cardinality of their intersection. The resulting value reflects the degree of overlap between the sets, indicating their similarity.

Generative model kernel Kernel functions can also be defined using probability distributions. In the context of generative models, where $\mathrm{P}(\mathbf{x})$ represents the probability distribution, a kernel function can be defined as:

$$k(\mathbf{x}, \mathbf{x}') = \mathrm{P}(\mathbf{x})\mathrm{P}(\mathbf{x}')$$

This kernel function is valid as it corresponds to the inner product in a one-dimensional feature space obtained by mapping $\mathbf{x}$ to $\mathrm{P}(\mathbf{x})$. It effectively measures the similarity between two samples by considering their respective probabilities under the generative model.

2.4.2 Kernel ridge regression

The loss function utilized in ridge regression is given by:

$$L(\mathbf{w}) = \frac{1}{2}(\mathbf{t} - \mathbf{\Phi w})^T (\mathbf{t} - \mathbf{\Phi w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

To solve it, we equate the gradient of L with respect to $\mathbf{w}$ to zero:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = \lambda\mathbf{w} - \mathbf{\Phi}^T(\mathbf{t} - \mathbf{\Phi w}) = 0$$

Now, instead of solving it for $\mathbf{w}$, let's perform a variable change ($\mathbf{a} = \lambda^{-1}(\mathbf{t} - \mathbf{\Phi w})$):

$$\mathbf{w} = \mathbf{\Phi}^T\lambda^{-1}(\mathbf{t} - \mathbf{\Phi w}) = \mathbf{\Phi}^T\mathbf{a}$$

Substituting $\mathbf{w}$ in the gradient, we have:

$$\begin{aligned}
&\lambda\mathbf{w} - \mathbf{\Phi}^T(\mathbf{t} - \mathbf{\Phi w}) = 0 \rightarrow \\
&\mathbf{\Phi}^T\left(\lambda\mathbf{a} - \left(\mathbf{t} - \mathbf{\Phi}\mathbf{\Phi}^T\mathbf{a}\right)\right) = 0 \rightarrow \\
&\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{a} + \lambda\mathbf{a} = \mathbf{t} \rightarrow \\
&\mathbf{a} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{t}
\end{aligned}$$

Here, $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T$ is known as the Gram matrix. The Gram matrix is an $N \times N$ matrix where each element represents the inner product between the feature vectors:

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}$$

The Gram matrix signifies the similarities between each pair of samples in the training data.

Prediction function To compute the prediction using the dual representation, we can utilize the following formula:

$$y(\mathbf{x}) = \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}) = \mathbf{a}\mathbf{\Phi}\boldsymbol{\phi}(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{t}$$

Here,$\mathbf{k}(\mathbf{x})$ is defined such that $k_n(\mathbf{x}) = k(\mathbf{x}_n, \mathbf{x})$ for all $\mathbf{x}_n \in \mathcal{D}$. Accordingly, the prediction is computed as the linear combination of the target values of the samples in the training set.

Comparison The original representation:

- Involves computing the inverse of $(\mathbf{\Phi}\mathbf{\Phi}^T + \lambda\mathbf{I}_M)$, which yields an $M \times M$ matrix.
- Is computationally convenient when M is relatively small.

The dual representation:

- Requires computing the inverse of $(\mathbf{K} + \lambda\mathbf{I}_N)$, which results in an $N \times N$ matrix.
- Is computationally favorable when N is very large or even infinite.
- Eliminates the need to explicitly compute $\mathbf{\Phi}$, enabling application to diverse data types such as graphs, sets, strings, and text.
- The computation of the similarity between data samples (i.e., the kernel function) is typically more efficient and simpler than calculating $\mathbf{\Phi}$.

2.4.3 Kernel regression

The k-nearest neighbors algorithm can be utilized for regression tasks by computing the average of the target values of the k nearest samples in the training data. This can be expressed as:

$$\hat{f}(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{x}_i \in N_k(\mathbf{x})} t_i$$

Nadaraya-Watson model In k-NN regression, the model output often exhibits significant noise due to the discontinuity of neighborhood averages. The Nadaraya-Watson model, also known as kernel regression, addresses this issue by employing a kernel function to calculate a weighted average of samples:

$$\hat{f}(\mathbf{x}) = \frac{\sum_{i=1}^{N} k(\mathbf{x}, \mathbf{x}_i) t_i}{\sum_{i=1}^{N} k(\mathbf{x}, \mathbf{x}_i)}$$

Typically, kernels are chosen based on their properties. Two common choices for kernels are:

- Epanechnikov Kernel (bounded support):

$$k(u) = \frac{3}{4}\left(1 - u^2\right) \qquad |u| \leq 1$$

- Gaussian Kernel (infinite support):

$$K(u) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{u^2}{2\sigma^2}}$$

2.4.4 Gaussian processes

Starting from the assumptions of Bayesian linear regression:

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$$

with the following prior probability:

$$\mathrm{P}(\mathbf{w}) = \mathcal{N}\left(\mathbf{w} | \mathbf{0}, \tau^2 \mathbf{I}\right)$$

Now, let's compute the prior distribution of the outputs of the regression function:

$$\mathbf{y} = \boldsymbol{\Phi}\mathbf{w} \implies \mathcal{N}(\mathbf{y} | \boldsymbol{\mu}, \mathbf{S})$$

Here:

- $\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{y}] = \boldsymbol{\Phi}\mathbb{E}[\boldsymbol{w}] = \mathbf{0}$
- $\mathbf{S} = \mathrm{Cov}(\mathbf{y}\mathbf{y}^T) = \boldsymbol{\Phi}\mathbb{E}[\boldsymbol{w}\boldsymbol{w}^T]\boldsymbol{\Phi}^T = \tau^2\boldsymbol{\Phi}\boldsymbol{\Phi}^T = \mathbf{K}$

In general, a Gaussian Process is defined as a probability distribution over a function $y(\mathbf{x})$ such that the set of values $y(\mathbf{x}_i)$ — for an arbitrary $\mathbf{x}_i$ — jointly have a Gaussian distribution. In our case:

$$\mathrm{P}(\mathbf{y}) = \mathcal{N}\left(\mathbf{y} | \mathbf{0}, \mathbf{K}\right)$$

where $\mathbf{K}$ is the Gram matrix defined as:

$$K_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = \tau^2 \boldsymbol{\phi}(\mathbf{x}_n)^T \boldsymbol{\phi}(\mathbf{x}_m)$$

This provides a probabilistic interpretation of the Kernel function as:

$$k(\mathbf{x}_n, \mathbf{x}_m) = \mathbb{E}\left[y(\mathbf{x}_n), y(\mathbf{x}_m)\right]$$

We can apply the usual approaches to design the kernels. Two families of kernels typically used with Gaussian processes are:

- Gaussian kernel:
$$k(\mathbf{x}, \mathbf{x}') = e^{-\frac{\|\mathbf{x}-\mathbf{x}'\|_2^2}{2\sigma^2}}$$
- Exponential kernel:
$$k(\mathbf{x}, \mathbf{x}') = e^{-\theta|\mathbf{x}-\mathbf{x}'|}$$

2.5 Support Vector Machines

CHAPTER 3

Model evaluation

3.1 Bias-variance framework

The bias-variance framework provides a structured approach for evaluating model performance.

Definition (*Data*)**.** Data are described as:

$$t_i = f(\mathbf{x}_i) + \varepsilon$$

where $\mathbb{E}[\varepsilon] = 0$ and $\mathrm{Var}[\varepsilon] = \sigma^2$.

Definition (*Model*)**.** The model is represented as:

$$\hat{t}_i = y(\mathbf{x}_i) + \varepsilon$$

learned from a sampled dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$.

Definition (*Performance*)**.** Performance is quantified by:

$$\mathbb{E}\left[(t - y(\mathbf{x}))^2\right]$$

which measures the expected squared error.

Hence, the expected squared error can be decomposed as follows:

$$\begin{aligned}
\mathbb{E}\left[(t - y(\mathbf{x}))^2\right] &= \mathbb{E}\left[\left(t^2 + y(\mathbf{x})^2 - 2ty(\mathbf{x})\right)\right] \\
&= \mathbb{E}\left[t^2\right] + \mathbb{E}\left[y(\mathbf{x})^2\right] - \mathbb{E}\left[2ty(\mathbf{x})\right] \\
&= \mathbb{E}\left[t^2\right] + \mathbb{E}[t]^2 - \mathbb{E}[t]^2 + \mathbb{E}\left[y(\mathbf{x})^2\right] + \mathbb{E}[y(\mathbf{x})]^2 - \mathbb{E}[y(\mathbf{x})]^2 - 2f(\mathbf{x})\mathbb{E}\left[y(\mathbf{x})\right] \\
&= \mathrm{Var}\left[t\right] + \mathbb{E}[t]^2 + \mathrm{Var}\left[y(\mathbf{x})\right] + \mathbb{E}[y(\mathbf{x})]^2 - 2f(\mathbf{x})\mathbb{E}\left[y(\mathbf{x})\right] \\
&= \mathrm{Var}\left[t\right] + \mathrm{Var}\left[y(\mathbf{x})\right] + \left(f(\mathbf{x}) - \mathbb{E}\left[y(\mathbf{x})\right]\right)^2 \\
&= \underbrace{\mathrm{Var}\left[t\right]}_{\sigma^2} + \underbrace{\mathrm{Var}\left[y(\mathbf{x})\right]}_{\text{variance}} + \underbrace{\mathbb{E}[f(\mathbf{x}) - y(\mathbf{x})]^2}_{\text{squared bias}}
\end{aligned}$$

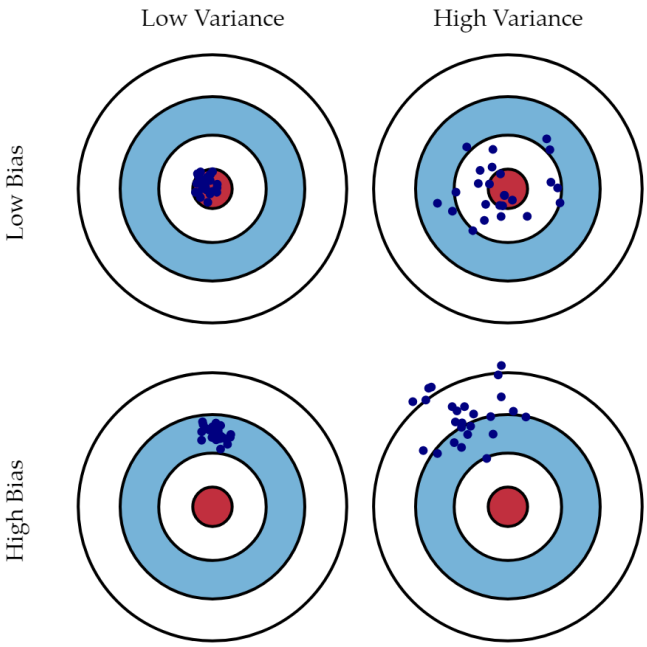


Figure 3.1: Bias-variance framework

Model variance When we sample multiple datasets $\mathcal{D}$, we obtain distinct models $y(\mathbf{x})$. Variance quantifies the dissimilarity between each model learned from a specific dataset and our anticipated learning outcome:

$$\text{variance} = \int \mathbb{E}\left[(y(\mathbf{x} - \bar{y}(\mathbf{x})))^2\right] \mathrm{P}(\mathbf{x}) d\mathbf{x}$$

The variance diminishes by simplifying the model or increasing the sample size.

Model bias Bias gauges the disparity between the truth (f) and our expected learning outcome ($\mathbb{E}\left[y(\mathbf{x})\right]$):

$$\text{bias}^2 = \int (f(\mathbf{x}) - \bar{y}(\mathbf{x}))^2 \mathrm{P}(\mathbf{x}) d\mathbf{x}$$

Bias decreases with more complex models.

Definition (*Data noise*)**.** Data noise (σ^2) represents the variance of data and remains constant regardless of data sampling or model complexity.

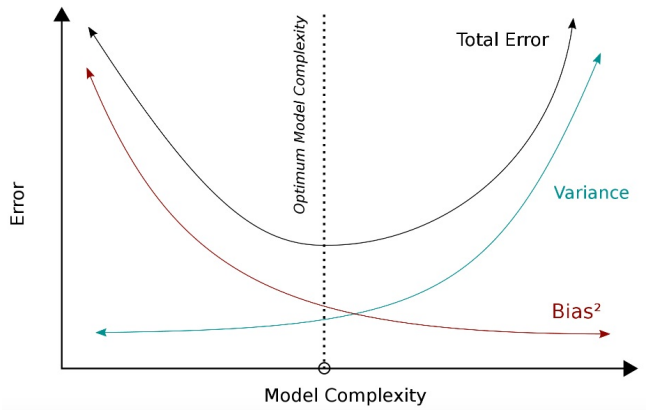
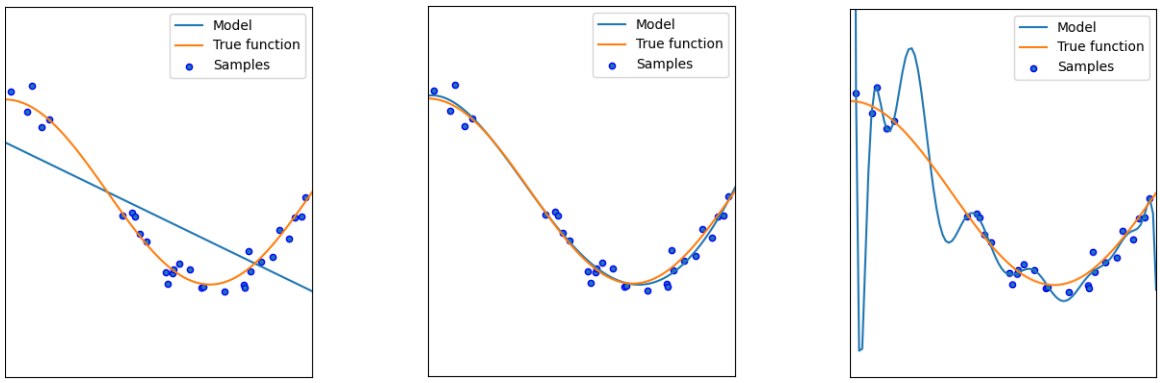


Figure 3.2: Bias-variance framework impact

In practical terms, the estimation is affected as follows:

- High variance leads to overfitting.
- High bias results in underfitting.
- Low bias and low variance yield a well-balanced approximation.



(a) High bias (b) Balanced (c) High variance

Figure 3.3: Bias-variance balancing

3.1.1 Regularization and bias-variance

The bias-variance decomposition elucidates why regularization enhances error reduction on unseen data. Lasso surpasses ridge regression when only a few features are linked to the output.

3.2 Model assessment

Given a dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$ with $i = 1, \ldots, N$, we can choose a model based on the computed loss L on $\mathcal{D}$. For regression, the loss function is defined as:

$$L_{train} = \frac{1}{N} \sum_{n=1}^{N} (t_n - y(\mathbf{x}_n))^2$$

And for classification, the loss function becomes:

$$L_{train} = \frac{1}{N} \sum_{n=1}^{N} I(t_n \neq y(\mathbf{x}_n))$$

The training error decreases as the model complexity increases.

However, it's important to note that the training error doesn't give an accurate estimate of the error on new data, known as the prediction error. For regression, the prediction error is represented as:

$$L_{true} = \iint (t - y(\mathbf{x}))^2 \mathrm{P}(\mathbf{x}, t) d\mathbf{x} dt$$

And for classification, it is:

$$L_{true} = \iint I(t \neq y(\mathbf{x}))\mathrm{P}(\mathbf{x}, t)d\mathbf{x}dt$$

Unfortunately, modeling the joint probability distribution $\mathrm{P}(\mathbf{x}, t)$ is often not feasible.

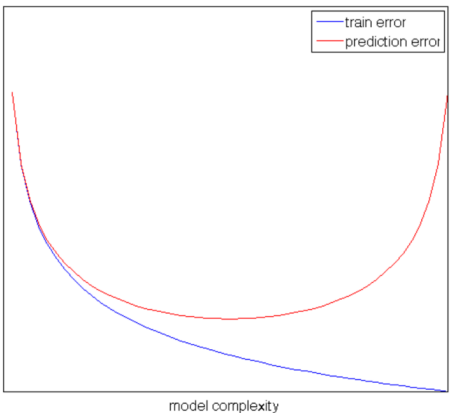


Figure 3.4: Train error compared to prediction error

Practical application In practical scenarios, data is typically randomly split into a training set and a test set. Model parameters are optimized using the training set, and the prediction error is estimated using the test set. For regression, this estimation yields:

$$L_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} (t_n - y(\mathbf{x}_n))^2$$

And for classification:

$$L_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} I(t_n \neq y(\mathbf{x}_n))$$

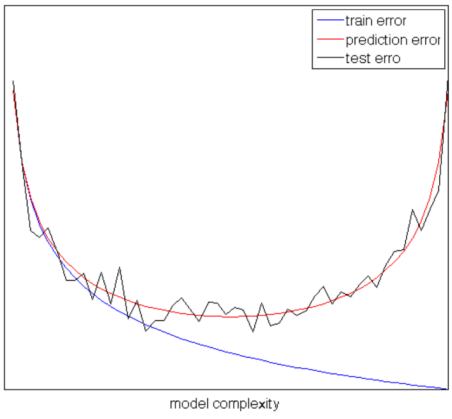


Figure 3.5: Error in practice

As the number of data points increases, these errors tend to converge, as depicted in the following figure:

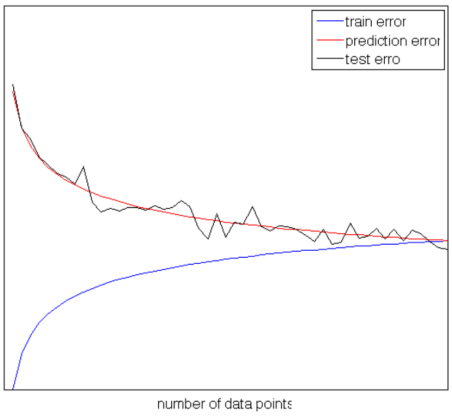


Figure 3.6: Error in function of the number of data points

Analyzing the train-test errors helps identify potential issues:

- *High bias*: when both training and test errors are higher than expected and close to each other.
- *High variance*: when the training error is significantly lower than expected and gradually approaches the test error.

Problems Frequently, data availability is constrained, and the test error tends to be minimal, leading to potential overestimation or underestimation of prediction error. Utilizing test error for model selection can result in overfitting to the test set. An unbiased estimate of prediction error is achievable only if the test set remains separate from both training and model selection phases.

3.2.1 Optimal model

To select the optimal model and determine the appropriate hyperparameters, we initially partition the data into three subsets: training data, validation data, and test data. The process involves the following steps:

1. Utilize the training data to train the model parameters.
2. For each trained model, assess its performance using the validation data to compute the validation error.
3. Identify the model with the lowest validation error, and subsequently, employ the test data to estimate the prediction error.

However, for this approach to be dependable, it's imperative that the validation data set is sufficiently sizable, especially in comparison to the training data set. Otherwise, there's a risk of overfitting to the validation data, potentially leading to the selection of a suboptimal model.

Leave-one-out cross validation Leave-one-out cross-validation (LOO-CV) involves training the model on all samples in the dataset $\mathcal{D}$ except for a single sample $\{\mathbf{x}_i, t_i\}$, and then

evaluating the model's performance on that omitted sample. The prediction error estimate of our model is then computed as the average error across all single-sample evaluations:

$$L_{LOO} = \frac{1}{N} \sum_{i=1}^{N} \left(t_i - y_{\mathcal{D}_i}(\mathbf{x}_i)\right)^2$$

Here, $y_{\mathcal{D}_i}$ represents the model trained on $\mathcal{D}$ excluding $\{\mathbf{x}_i, t_i\}$.

The L_{LOO} estimate of prediction error provides an almost unbiased assessment (slightly pessimistic). However, LOO-CV is computationally intensive due to its requirement to repeatedly train models on nearly all data points.

K-fold cross validation K-fold cross-validation involves randomly dividing the training data $\mathcal{D}$ into k folds: $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$. For each fold $\mathcal{D}_i$, the model is trained on $\mathcal{D}$ excluding $\mathcal{D}_i$, and then the error is computed on $\mathcal{D}_i$ as follows:

$$L_{\mathcal{D}_i} = \frac{k}{N} \sum_{(\mathbf{x}_n, t_n) \in \mathcal{D}_i} \left(t_n - y_{\mathcal{D}\setminus\{\mathcal{D}_i\}}(\mathbf{x}_n)\right)^2$$

Finally, the prediction error is estimated as the average error computed across all folds:

$$L_{k-fold} = \frac{1}{k} \sum_{i=1}^{k} L_{\mathcal{D}_i}$$

The $L_{k\text{-fold}}$ estimate of prediction error provides a slightly biased (pessimistic) assessment but is computationally less expensive. Typically, k is set to ten.

Other metrics Various metrics are employed to evaluate models by adjusting their training error based on their complexity:

- Mallows's C_p:

$$C_p = \frac{1}{N}\left(\text{RSS} + 2M\sigma^2\right)$$

- Akaike Information Criteria:

$$\text{AIC} = -2\ln(L) + 2M$$

- Bayesian Information Criteria:

$$\text{BIC} = -2\ln(L) + M\ln(N)$$

- Adjusted R^2:

$$A_{R^2} = 1 - \frac{\text{RSS}/(n-m-1)}{\text{TSS}/(N-1)}$$

Here, M represents the number of parameters, N denotes the number of samples, L signifies the loss function, σ^2 stands for the estimate of noise variance, RSS corresponds to the residual sum of squares, and TSS indicates the total sum of squares.

AIC and BIC are typically utilized when maximizing the log-likelihood. BIC tends to penalize model complexity more severely compared to AIC.

3.3 Model complexity

Introducing an additional feature leads to an exponential growth in the volume of the input space. This growth leads to the following problems:

- *Computational cost*: the computational resources required to process and analyze the expanded input space increase significantly.
- *Data quantity*: the amount of data needed to effectively explore and train models in the expanded input space may be substantial.
- *Large model variance* (overfitting): with the increased complexity of the input space, there is a higher risk of models capturing noise or irrelevant patterns, leading to overfitting and decreased generalization performance.

Our goal is to choose the model with the minimal prediction error, which can be attained by decreasing the variance of the model:

- *Feature selection*: by carefully designing the feature space, we can choose the most impactful subset from all available features.
- *Dimensionality reduction*: mapping the input space to a lower-dimensional representation can effectively reduce complexity and variance.
- *Regularization*: shrinkage of parameter values towards zero helps control model complexity and mitigate overfitting.

These approaches are not mutually exclusive and can be combined to enhance model performance.

3.3.1 Feature selection

The most straightforward approach appears to be comparing all possible combinations of features. Given M features, for each $k = 1, \ldots, M$, we would need to train all $\binom{M}{k} = \frac{M!}{k!(M-k)!}$ models with exactly k features and select the optimal one. However, this procedure quickly becomes computationally impractical.

In practical scenarios, feature selection is often carried out based on the specific model being utilized:

- *Filter*: features are assessed individually using certain evaluation metrics (e.g., correlation, variance, information gain), and the top k features are selected. While this method is very fast, it fails to capture any subset of mutually dependent features.
- *Embedded*: feature selection is integrated into the machine learning approach itself (e.g., lasso, decision trees). Although this method is not computationally expensive, the features identified are specific to the chosen learning approach.
- *Wrapper*: a search algorithm is employed to identify a subset of features by iteratively training a model with different feature subsets and evaluating their performance. This method utilizes either a simpler model or a basic machine learning approach to evaluate the features. Greedy algorithms are typically employed to search for the best feature subset.

3.3.2 Dimensionality reduction

Dimensionality reduction aims to decrease the dimensions of the input space, but it differs from feature selection in two significant ways:

- It utilizes all features and transforms them into a lower-dimensional space.
- It is an unsupervised approach, meaning it doesn't rely on labeled data for training.

There are numerous methods for performing dimensionality reduction, including:

- Principal Component Analysis (PCA).
- Independent Component Analysis (ICA).
- Self-Organizing Maps.
- Autoencoders.
- ISOMAP.
- t-SNE.

3.4 Ensemble

We've explored methods to decrease variance while balancing increased bias. However, we want to reduce variance without amplifying bias or mitigate bias altogether.

These objectives can indeed be achieved through the utilization of two ensemble methods involving the learning of multiple models and their combination:

- *Bagging*: involves training multiple models independently on different subsets of the data and then combining their predictions.
- *Boosting*: utilizes an iterative approach where models are sequentially trained, each aiming to correct the errors of its predecessors, leading to the creation of a strong ensemble model.

3.4.1 Bagging

Let assume to have N datasets and to learn from them N models, $y_1, y_2, \ldots, y_N$. Now let us compute an aggregate model as:

$$y_{AGG} = \frac{1}{N} \sum_{i=1}^{N} y_i$$

If the datasets are independent, the model variance of y_{AGG} will be $\frac{1}{N}$ of the model variance of the single model y_i. However, we generally do not have N datasets.

Bagging, short for Bootstrap Aggregation, involves the following steps:

1. Generate N datasets by applying random sampling with replacement.
2. Train a model (classification or regression) using each dataset generated.

3. To predict new samples, apply all the trained models and combine their outputs using majority voting (for classification) or averaging (for regression).

Bagging is generally beneficial as it reduces variance, although the sampled datasets are not independent. It proves particularly useful with unstable learners, characterized by significant changes with small dataset variations (low bias and high variance), and in scenarios with a high degree of overfitting (low bias and high variance). However, it does not offer much help with robust learners, which are insensitive to data changes (typically higher bias but lower variance).

3.4.2 Boosting

Boosting aims to minimize bias by employing a series of simple (weak) learners.

The core concept of boosting involves iteratively training a sequence of weak learners, with each iteration concentrating on the samples misclassified in the preceding iteration.

Ultimately, an ensemble model is constructed by combining the outputs of all the weak learners trained.

3.4.3 Summary

The characteristics of bagging are:

- Decreases variance.
- Less effective for stable learners.
- Applicable with noisy data.
- Generally beneficial, though the improvement might be modest.
- Naturally suited for parallelization.

The characteristics of boosting are:

- Reduces bias (typically without overfitting).
- Compatible with stable learners.
- May encounter challenges with noisy data.
- Not always effective, but can yield significant improvements.
- Sequential in nature.