STA314H1: Statistical Methods for Machine Learning I

Haoran Yu

April 8, 2025

Contents

1 Introduction

Statistical learning is a subfield of statistics that focuses on understanding the relationship between variables in data. The primary goal is often inference—understanding how a set of inputs (predictors) affects an output (response), and evaluate the algorithm using statistics.

Machine learning originated in computer science and is more focused on prediction accuracy and automation. The primary goal is often making predictions or decisions based on data without necessarily understanding the underlying relationships. It emphasizes learning from data, especially large datasets.

Thus, statistical learning is more concerned with statistical inference and understanding relationships between variables. Machine learning focuses on prediction accuracy and often works as a black box where model interpretability is secondary. So, Statistical learning is a subset of the broader field of machine learning.

Both fields rely heavily on core mathematical disciplines like calculus, probability theory, and linear algebra. Many algorithms used in both ML and SL overlap, as they are built on these foundational principles. However, the difference lies in their respective goals: while ML is more concerned with optimizing algorithms for better computational performance and prediction accuracy, SL is focused on understanding the models themselves and evaluating their reliability in terms of statistical inference.

The course focuses on Statistical Learning due to its broader relevance in fields such as biology, neuroscience, medicine, and finance. One of the key takeaways is that an understanding of SL is essential for grasping more complex algorithms, like those used in neural networks, which are pivotal in ML. The course aims to impart the fundamental principles behind various techniques, allowing you to not only use them but also know when and why to apply them, making you better equipped for roles such as a data scientist or quantitative analyst. Statistical learning, being foundational, serves as a crucial stepping stone to mastering more advanced methods and algorithms.

2 Linear algebra and probability review

2.1 Matrix algebra and linear algebra

Definition 2.1: Scalar

A scalar is the simplest form of data, representing a single number, which could belong to various numerical sets like integers, real numbers, or rational numbers. Scalars are often represented using italic fonts, such as a, n, or x. Scalars serve as basic building blocks in more complex structures like vectors and matrices.

Definition 2.2: Vectors

A vector is an array of numbers, with the dimension d, representing its size. Vectors can contain various types of elements such as integers, real numbers, or binary values. In notation, vectors are typically written in bold (e.g., $\boldsymbol{x}$), and their size and type can be indicated, for example, $\boldsymbol{x} \in \mathbb{R}^d$ denotes a vector of real numbers with d dimensions. A common way to express a vector is:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

Definition 2.3: Matrix

A matrix is an array of numbers with two indices, commonly denoted by $A_{i,j}$, where i represents rows and j represents columns. Just like vectors, matrices can consist of different types of numbers (real, integer, binary). A matrix with m rows and n columns is represented as $A \in \mathbb{R}^{m \times n}$. A sample matrix could be:

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}$$

Definition 2.4: Matrix (Dot) Product

The matrix product of two matrices A and B, denoted AB, is defined by the sum of element-wise products across rows of A and columns of B :

$$(AB)_{i,j} = \sum_k A_{i,k} B_{k,j}$$

This formula also applies to matrix-vector products, such as $A\boldsymbol{x}$, where $\boldsymbol{x}$ is a vector.

Definition 2.5: Identity matrix

The identity matrix, denoted I_d, is a special square matrix where all diagonal elements are 1 , and all off-diagonal elements are 0 . For example, the 3×3 identity matrix is:

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

When a vector $\boldsymbol{x} \in \mathbb{R}^d$ is multiplied by the identity matrix, the result is the original vector: $I_d \boldsymbol{x} = \boldsymbol{x}$.

Definition 2.6: Matrix transpose

The transpose of a matrix A, denoted $A^\top$, flips the matrix over its diagonal. In other words, the element at position (i, j) in A becomes the element at (j, i) in $A^\top$. A matrix A and its transpose $A^\top$ are related by the property:

$$(AB)^\top = B^\top A^\top$$

This means the order of multiplication reverses when transposing a matrix product.

Definition 2.7: Systems of equations

A system of linear equations can be represented in matrix form as $A\boldsymbol{x} = \boldsymbol{b}$, where A is the coefficient matrix, $\boldsymbol{x}$ is the vector of variables, and $\boldsymbol{b}$ is the vector of constants. Expanding this, the matrix equation becomes:

$$A_{1,1}x_1 + A_{1,2}x_2 + \cdots + A_{1,n}x_n = b_1$$

Definition 2.8: Matrix inversion

The matrix inverse A^{-1} is the matrix such that $A^{-1}A = I_d$. If a matrix is invertible, you can solve the linear system $A\boldsymbol{x} = \boldsymbol{b}$ by multiplying both sides by A^{-1} :

$$A^{-1}A\boldsymbol{x} = A^{-1}\boldsymbol{b} \quad \Rightarrow \quad I_d\boldsymbol{x} = A^{-1}\boldsymbol{b}$$

However, matrix inversion is not always possible.

Remark.

A matrix is not always invertible. Common reasons include:

- More rows than columns
- More columns than rows
- Linearly dependent rows or columns

Remark.

A system of equations may have no solution, many solutions, or exactly one solution. The system is invertible (i.e., has a unique solution) if the matrix A is invertible.

Definition 2.9: Norms

A norm is a function that measures the size or magnitude of a vector. Norms obey certain properties, including the triangle inequality. The most common norms include:

- L^2-norm (Euclidean norm):

$$\|\boldsymbol{x}\|_2 = \left(\sum_i |x_i|^2\right)^{1/2}$$

- L^1-norm:

$$\|\boldsymbol{x}\|_1 = \sum_i |x_i|$$

- Max norm (infinity norm):

$$\|\boldsymbol{x}\|_\infty = \max_i |x_i|$$

Remark.

Here are some specical matrix and special vectors:

- **Unit vector:** A vector with L^2-norm equal to 1: $\|\boldsymbol{x}\|_2 = 1$.
- **Symmetric matrix:** A matrix that is equal to its transpose: $A = A^\top$.
- **Orthogonal matrix:** A matrix whose transpose is also its inverse: $A^\top A = AA^\top = I_d$.

Definition 2.10: Trace

The trace of a square matrix A is the sum of its diagonal elements:

$$\mathrm{Tr}(A) = \sum_i A_{i,i}$$

It has useful commutative properties, such as:

$$\mathrm{Tr}(ABC) = \mathrm{Tr}(CAB) = \mathrm{Tr}(BCA)$$

2.2 Probability and statistics

Remark.

In probability and statistics, random variables are denoted by capital letters (e.g., X and Y).

Remark.

The parameters such as $\beta_1, \ldots, \beta_p$ or the function $f : \mathcal{X} \to \mathcal{Y}$ are treated as deterministic (non-random). Of course, being Bayesian is an exception.
The data points (x_i, y_i) for $1 \le i \le n$ are actual values, observed in practice. They can be thought as the realizations of random variables (X_i, Y_i) for $1 \le i \le n$.

Definition 2.11: Variance

Let X and Y be two random variables.

$$\mathrm{Var}(X) = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}\left[X^2\right] - (\mathbb{E}[X])^2$$

More generally, for any function f,

$$\mathrm{Var}(f(X)) = \mathbb{E}\left[(f(X) - \mathbb{E}[f(X)])^2\right] = \mathbb{E}\left[(f(X))^2\right] - (\mathbb{E}[f(X)])^2$$

Definition 2.12: Expectation

For any function f and g, if X is independent of Y, then

$$\mathbb{E}[f(X)g(Y)] = \mathbb{E}[f(X)]\mathbb{E}[g(Y)]$$

and

$$\mathbb{E}[f(X) \mid Y] = \mathbb{E}[f(X)]$$

For any function h,

$$\begin{aligned}\mathbb{E}[h(X,Y)] &= \mathbb{E}_X\left[\mathbb{E}_{Y|X}[h(X,Y) \mid X]\right] \\ &= \mathbb{E}_Y\left[\mathbb{E}_{X|Y}[h(X,Y) \mid Y]\right]\end{aligned}$$

where $\mathbb{E}_X$ is the expectation w.r.t. the randomness of X whereas $\mathbb{E}_{Y|X}$ is w.r.t. the randomness of $Y \mid X$.

Remark.

X is said to be uncorrelated with Y if

$$\mathrm{Cov}(X,Y) = 0$$

In particular, the fact that X is independent of Y implies that $\mathrm{Cov}(X,Y) = 0$

Remark.

For any constants a, b,

$$\mathrm{Var}(aX+bY) = a^2\,\mathrm{Var}(X) + b^2\,\mathrm{Var}(Y) + 2ab\,\mathrm{Cov}(X,Y)$$

In particular, if X is uncorrelated with Y, then

$$\mathrm{Var}(aX+bY) = a^2\,\mathrm{Var}(X) + b^2\,\mathrm{Var}(Y)$$

3 Introduction to statistical learning as a subset of machine learning

Machine learning (ML) is a methodology where algorithms are programmed to autonomously learn from data or experiences, without being explicitly programmed for specific tasks. This approach is valuable for several reasons:

1. **Complex Problems:** Some tasks, such as vision or speech recognition, are exceedingly complex and cannot be effectively solved by manual programming due to the intricacies involved.
2. **Adaptive Systems:** Machine learning is crucial for applications needing adaptation to changing conditions, such as spam detection in emails.
3. **Performance Improvement:** ML systems can often achieve higher accuracy and efficiency than human-crafted solutions because they can adapt and learn from vast amounts of data.

There are three types of learning algorithms in machine learning:

1. **Supervised learning** involves training a model on a labeled dataset, where each example in the dataset is composed of an input-output pair. Here, the model is tasked with learning a mapping from inputs to outputs, making it suitable for:
   - Regression problems: Where the output variable is a continuous value (e.g., predicting house prices).
   - Classification problems: Where the output variable is a category (e.g., distinguishing between spam and non-spam emails).
2. In **unsupervised learning**, the data used to train the model is not labeled, meaning that no explicit instructions are given to the algorithm on what to do with the data. Instead, the algorithm must learn the patterns and structures from the data itself.
3. **Reinforcement learning** is a type of machine learning where an agent learns to behave in an environment by performing actions and seeing the results. Unlike supervised learning, the agent is not told which actions to take, but instead must discover which actions yield the most reward by trying them.

In this course, we will cover mostly supervised learning, and some unsupervised learning, they all included many important methods.

Some important real life application of supervised learning include:

1. **Spam Detection:** For example, creating a model to filter spam emails involves analyzing features such as the frequency of specific words or phrases.
2. **Image Recognition:** Learning from labeled images to predict the content of new images, as demonstrated by the recognition of objects like cats or dogs in photographs.
3. **Handwritten Digit Recognition:** Identifying numbers from images, such as those used in ZIP codes.

AlphaGo is a prime example of reinforcement learning where the system learned to play Go at a professional level by training against itself and optimizing its strategies over millions of iterations.

There are also many other integration of machine learning in various fields:

- **Natural Language Processing (NLP):** Used for machine translation, sentiment analysis, and more.
- **Computer Vision:** Includes applications like object detection and semantic segmentation such as differentiate between various geographical features in satellite images.

The complexity in unsupervised learning lies in its lack of explicit labels which makes direct performance evaluation tricky. However, it's instrumental in understanding large datasets without predefined outcomes. So, they can be used to pre-process data used for supervised learning. For example, image segmentation is a computer vision technique that divides an image into multiple segments (sets of pixels) using k-mean clustering (a unsupervised algorithm). The goal is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images.

4 General concepts of supervised learning

4.1 What is supervised learning?

The core components of a supervised learning includes:

- **Outcome Measurement (Y):** Also known as the dependent variable, response, or target, it's what the model aims to predict or classify.
- **Predictor Measurements (X):** These are the independent variables or features that are used to predict Y. They are also called inputs, regressors, or covariates.

Using supervised learning, we wish to acomplish:

1. **Prediction:** To accurately forecast future instances of $\mathcal{Y}$.
2. **Estimation:** Understanding how changes in $\mathcal{X}$ affect Y.
3. **Model Selection:** Identifying the most effective model for prediction.
4. **Inference:** Evaluating the reliability and quality of predictions, estimations, and model selection.

We have training data $(x_1, y_1), \ldots, (x_n, y_n)$. These are observations (instances, realizations) of the measurement (X, Y).

1. features $x_1, \ldots, x_n \in X$ and corresponding
2. outcome $y_1, \ldots, y_n \in Y$.

The relationship between the input features X and the output Y is typically modeled as $Y = f(X) + \epsilon$. Here, $f(X)$ represents the underlying true relationship, and ϵ captures random errors and discrepancies not explained by our model. The fundamental aim in supervised learning is to approximate this true function f, based on the provided training data $\mathcal{D}^{\text{train}}$, which consists of independent and identically distributed (i.i.d.) samples of (X, Y).

So, our goal is to learn a predictor (function / mapping) $g : X \to Y$ such that for a new test data $x_* \in X$,

$$g(x_*) \approx y_*.$$

The motivations for estimating f are:

1. **Prediction:** For any new input $X = x$, the function $f(x)$ often serves as an excellent prediction of Y. It is essentially the best prediction we can make, minimizing expected loss under specific criteria.
2. **Feature Selection:** Understanding which components of the vector X (comprising multiple features $X_1, X_2, \ldots, X_p$) are relevant or irrelevant in explaining Y is crucial. For instance, if we have: $f(X_1, X_2, X_3) = 0.5 + 4X_1 + X_1^2 - 2X_2^3$ This function tells us how each feature contributes to the outcome, which is pivotal for feature selection and simplification of models.

Example.

> Consider a practical scenario where we have data for n products, with each product having associated features like the budget for TV, radio, and newspaper advertising. Suppose the true model is given by: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \epsilon$. This model helps to determine how changes in the advertising budget (across different media) impact sales.

Remark.

> Here are some remarks on the notations we use:
>
> - Let x_{ij} denote the value of the j th feature for observation i, where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, p$.
> - Let y_i denote the response variable for the i th observation.
> - Training data consist of $\mathcal{D}^{\text{train}} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where $x_i = (x_{i1}, \ldots, x_{ip})^T$.

4.1.1 Parametric and non-parametric methods

There are two categories of approaches to estimate f based on $\mathcal{D}^{\text{train}}$:

- **Parametric method:** These assume a specific functional form for f.
- **Non-parametric method:** These do not assume any specific form for f.

Example.

The linear model is an important example of a parametric model:

$$Y = f(X) + \epsilon, \quad \text{with} \quad f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$

Correspondingly, we would estimate f by

$$\hat{f}_{\text{linear}}(X) = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \cdots + \hat{\beta}_p X_p$$

constructing $\hat{f}_{\text{linear}}$ reduces the problem of estimating a function to that of estimating $(p+1)$ parameters $(\beta_0, \beta_1, \ldots, \beta_p)$. The procedure of obtaining $\hat{\beta}_0, \ldots, \hat{\beta}_p$ is called fitting, i.e. fitting the model to the training data.

Example.

If the linear predictor does a poor job for fitting the data, one can consider more complex forms of f, such as:

- Quadratic form: $f(X) = \beta_0 + \beta_1 X + \beta_2 X^2$
- Step-wise form: $f(X) = \beta_0 + \beta_1 1\{X \leq 0.5\} + \beta_2 1\{X > 0.5\}$.
- Polynomial form: $f(X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \cdots + \beta_d X^d$
- Two layer neural net: $f(X) = \sigma\left(W_1 \sigma\left(W_0 X + b_0\right) + b_1\right)$

Definition 4.1: Interpretation

The ability to understand how the predictors X contribute to predicting Y.

Remark.

You can keep adding complexity by considering more complicated f. However, f gets less interpretable as its form gets more complicated. For example, linear models are easy to interpret, neural nets are not.

Example.

The nearest neighborhood method is a example of non-parametric method. To predict at $X = x$ with $\mathcal{N}(x)$ being some neighborhood of x,

$$\hat{f}(x) = \frac{1}{|\mathcal{N}(x)|} \sum_{i: x_i \in \mathcal{N}(x)} y_i$$

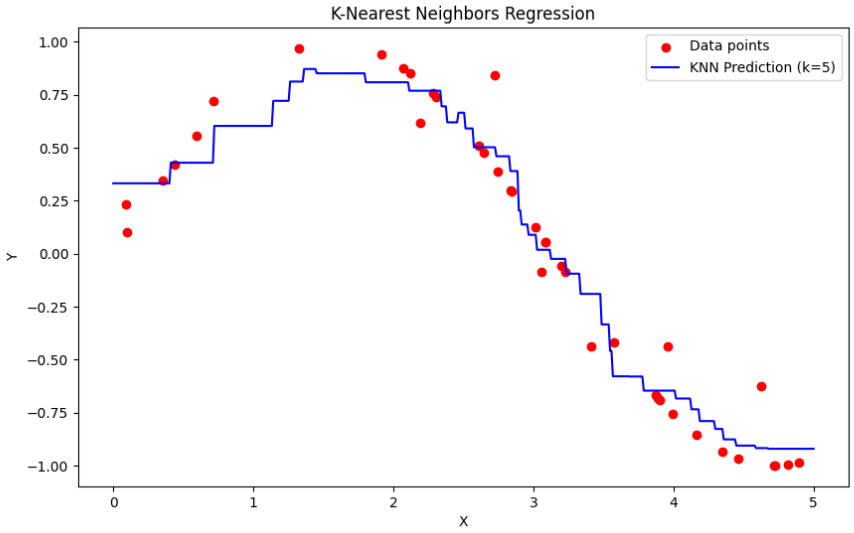

Figure 1: 5 - Nearest Neighbors Regression

Example.

There are more complex version of the neighborhood method, such as kernel estimator or spline estimator.

Pros and Cons of nearest neighbors method

Pros:

1. Flexibility in model form, meaning the model can adapt to any underlying data distribution without any assumption about the function's shape.
2. perform exceptionally well if the number of predictors p is small, i.e $p \leq 4$ relative to the number of data points n. This effectiveness is due to the denser sampling of the input space, which improves the likelihood that the neighbors are truly similar to the query point.

Cons:

1. As the number of features p increases, the volume of the input space grows exponentially, and the available data become sparse. This sparsity is problematic as the nearest neighbors need to be very close to the query point to make accurate predictions. However, in high dimensions, all points tend to be far away from each other, leading to less reliable and significantly more variable predictions. This is called Curse of Dimensionality.

4.1.2 Model selection process in general

The process of selecting the best predictive model involves comparing different types of models:

- **Parametric vs. Parametric:** Comparing different parametric models to determine which fits the data best according to predefined criteria.
- **Parametric vs. Non-parametric:** Weighing the benefits of a theoretically informed model (parametric) against a more flexible, data-driven approach (non-parametric).
- **Non-parametric vs. Non-parametric:** Choosing between different non-parametric models based on their performance metrics.

The selection criteria must be systematic and objective, ensuring the chosen model $\hat{f}$ best captures the underlying pattern without fitting noise or unnecessary complexity.

5 Regression model as supervised learning algorithm

Here, we will explore the regression model as one of the algorithm for supervised learning. A regression model can be parametric or non-parametric. Linear regression is a parametric regression model, while neighborhood algorithm is a non-parametric regression model. We will also introduce metrics that can help us evaluate different models.

5.0.1 Metrics for evaluating $\hat{f}$ with regression model

The Mean Squared Error (MSE) is a fundamental metric used in regression to evaluate model performance. It measures the average of the squares of the errors—the differences between observed values and values predicted by the model.

Ideally, we'd assess a model's MSE as $\mathbb{E}\left[\left(Y_* - \hat{f}\left(X_*\right)\right)^2\right]$. Here, (X_*, Y_*) is a new data pair not in the training set $\mathcal{D}^{\text{train}}$, and the expectation is over the distribution of new data pairs as well as the randomness in the model $\hat{f}$ due to the training data used to fit it.

Since the theoretical MSE cannot be directly computed (as the true distribution of (X_*, Y_*) and ϵ are unknown), we use the training and test data to estimate it:

1. **Training MSE:** $\mathrm{MSE}(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}\left(x_i\right)\right)^2$ This metric assesses how well the model fits the training data but can be misleading as it may favor overly complex models that overfit the data.

2. **Test MSE:** $\mathrm{MSE}_T(\hat{f}) = \frac{1}{m}\sum_{i=1}^{m}\left(y_{T_i} - \hat{f}\left(x_{T_i}\right)\right)^2$ Calculated using a separate test set $\mathcal{D}_{\text{test}}$, the test MSE provides a better measure of how well the model generalizes to new data. It's preferable to select the model that minimizes this metric.

Remark.

> If test data is available, we can directly compute $\mathrm{MSE}_T(\hat{f})$. Otherwise, we use a resampling technique called cross-validation, which will be discuess in detail later.

Remark.

> Here is a extra remark explaining why training MSE $\mathrm{MSE}(\hat{f}) := \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}\left(x_i\right)\right)^2$ always decreases as $\hat{f}$ gets more complex. This is because complex models have more degrees of freedom to adjust their parameters to closely fit a greater variety of patterns in the training data.

Remark.

> An ideal $\hat{f}$ should minimize the expected MSE. Minimizing the expected MSE means finding the model $\hat{f}$ that, on average, has the lowest possible squared difference between the predicted values and the actual values across all possible datasets.

5.0.2 Overfitting and underfitting for regression with MSE

Overfitting occurs when a model is too complex, capturing noise in the data as genuine relationships, leading to poor generalization to new data.

Conversely, **underfitting** happens when a model is too simple to capture the underlying patterns in the data, also resulting in poor performance on new data.

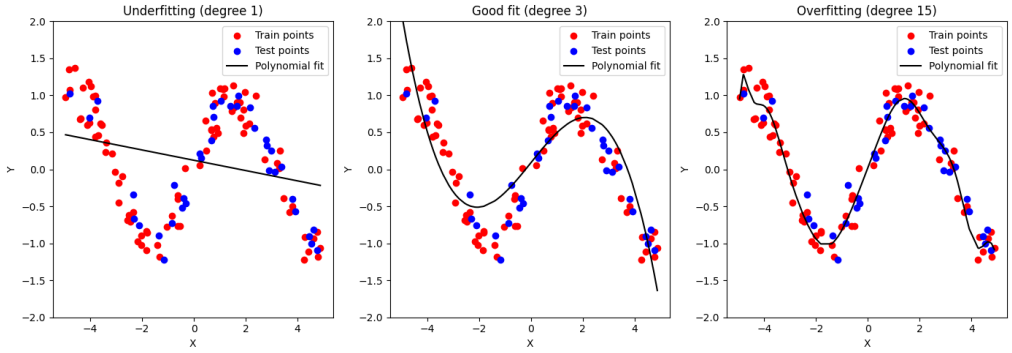

Figure 2: Underfitting, good fitting, and overfitting

5.0.3 Bias-Variance decomposition for regression

Suppose (X_*, Y_*) is a new observation pair, where $Y_* = f(X_*) + \epsilon_*$ and ϵ_* is a noise component with zero mean ($\mathbb{E}[\epsilon_*] = 0$) and some variance. The goal of our learning algorithm is to create an estimator $\hat{f}$ based on the training data $\mathcal{D}^{\text{train}}$, which predicts Y_* from X_*.

The mean squared error (MSE) of the estimator $\hat{f}$ at a new point $X_* = x_*$ is the expectation of the squared difference between the true value Y_* and the predicted value $\hat{f}(x_*)$, conditioned on $X_* = x_*$. Mathematically, this is expressed as $\mathbb{E}\left[\left(Y_* - \hat{f}(X_*)\right)^2 \mid X_* = x_*\right]$.

Plugging this into the MSE formula gives: $\mathbb{E}\left[\left(f(x_*) + \epsilon_* - \hat{f}(x_*)\right)^2 \mid X_* = x_*\right]$. Expanding the square in the expectation: $\mathbb{E}\left[\left(f(x_*) - \hat{f}(x_*)\right)^2 + 2\left(f(x_*) - \hat{f}(x_*)\right)\epsilon_* + \epsilon_*^2 \mid X_* = x_*\right]$

Since $\mathbb{E}[\epsilon_*] = 0$, the cross-term involving ϵ_* disappears, leaving: $\left(f(x_*) - \hat{f}(x_*)\right)^2 + \mathbb{E}[\epsilon_*^2]$. This can be further broken down into:

- The variance of the estimator $\hat{f}(x_*)$: $\text{Var}\left(\hat{f}(x_*)\right)$
- The squared bias of the estimator: $\left(\mathbb{E}\left[\hat{f}(x_*)\right] - f(x_*)\right)^2$
- The irreducible error, which is the variance of ϵ_* : $\text{Var}(\epsilon_*)$

Therefore, the full decomposition of the MSE becomes: $\text{Var}\left(\hat{f}(x_*)\right) + \left(\mathbb{E}\left[\hat{f}(x_*)\right] - f(x_*)\right)^2 + \text{Var}(\epsilon_*)$. So,

$$
\begin{aligned}
&\mathbb{E}\left[\left(Y_* - \hat{f}(X_*)\right)^2 \mid X_* = x_*\right] \\
&= \underbrace{\text{Var}\left(\hat{f}(x_*)\right)}_{\text{Variance}} + \underbrace{(\mathbb{E}\left[\hat{f}(x_*)\right] - f(x_*))^2}_{\text{Bias}} + \underbrace{\text{Var}(\epsilon_*)}_{\text{Irreducible error}} \\
&\geq \text{Var}(\epsilon_*)
\end{aligned}
$$

1. **Variance:** This part of the error comes from the variability of the model prediction $\hat{f}(x_*)$ around its mean. High variance indicates that the model is very sensitive to the specific set of training data.
2. **Bias:** This error measures how much $\hat{f}$ on average differs from the true function f. High bias can occur when the model is too simple to capture the underlying pattern.
3. **Irreducible Error:** This is the error inherent in the problem itself due to the randomness ϵ_* in the output. It cannot be reduced by any model.

Remark.

> The conditional MSE $\mathbb{E}\left[\left(Y_* - \hat{f}(X_*)\right)^2 \mid X_* = x_*\right]$ gives insight into the model's accuracy at a specific input $X_* = x_*$.

Remark.

The unconditional MSE provides a global view of the model's overall accuracy but doesn't capture how the model performs at individual points.

5.0.4 Bias-Variance trade-off

As the complexity or flexibility of an estimator $\hat{f}$ increases, it can adapt more closely to intricate patterns in the training data. Common transitions from less to more complex models include moving from linear regression (a parametric method) to nonparametric methods such as decision trees or neural networks.

More complex models tend to have higher variance because they fit the training data more closely but might vary significantly with slight changes in that data. As models become more complex, they generally become capable of representing the problem more accurately, thereby reducing bias.

The variance of a model is inversely proportional to the sample size. More data typically reduces the effect of overfitting, making complex models more viable. For small datasets, complex models may perform poorly due to high variance. For large datasets, complex models often perform better as their higher capacity can capture more subtle patterns in the data without as much overfitting.

Thus:

$$\text{Variance} \propto \frac{\text{Complexity of the model}}{n} \text{ and Bias} \propto \frac{1}{\text{Complexity of the model}}$$

Remark.

When choosing between models $\hat{f}_1$ and $\hat{f}_2$ that have similar expected mean squared errors (MSE), the preference usually goes to the more parsimonious (simpler or less complex) model. This preference is due to the principle of Occam's Razor, which suggests that simpler explanations are more likely to be correct than complex ones.

Remark.

An increase in variance does not always guarantee a decrease in bias, and vice versa.

5.0.5 Alternative metrics

There are alternative metrics for measuring $\hat{f}$, such as the Sum of Absolute Difference (SAM):

$$\mathbb{E}[|Y - \hat{f}(X)|], \quad \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{f}\left(x_i\right)\right|$$

5.0.6 A special case: regression with categorical or ordinal Y

When dealing with classification problems where the target variable Y is categorical or ordinal, the evaluation of predictive models $\hat{f}$ shifts from using continuous metrics like Mean Squared Error (MSE) to classification-specific metrics.

In classification tasks, the primary interest is often in the accuracy of categorical predictions rather than the magnitude of error. Thus, the **expected error rate** becomes a key metric. It is defined as:

$$\mathbb{E}[1\{Y_* \neq \hat{f}((X_*)\}]$$

Here, $1\{Y_* \neq \hat{f}(X_*)\}$ is an indicator function that returns 1 if the prediction $\hat{f}(X_*)$ does not match the actual value Y_*, and 0 otherwise. The expected error rate is thus the probability that the predictions do not match the true values, averaged over the distribution of the data.

The training error rate measures the proportion of misclassifications made by the model on the training dataset. It is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n} 1\left\{y_i \neq \hat{f}\left(x_i\right)\right\}$$

Each term in the sum is an instance of the indicator function, where y_i is the actual category of the i th sample and $\hat{f}(x_i)$ is the predicted category. If the prediction is incorrect, the function returns 1 ; otherwise, it returns 0 . The sum of these values is then divided by the total number of observations n in the training set, resulting in the fraction of misclassifications.

Similarly, **the test error rate** is calculated on a separate test dataset that was not used in training the model. It is defined as:

$$\frac{1}{m}\sum_{i=1}^{m} 1\left\{y_{Ti} \neq \hat{f}(x_{Ti})\right\}$$

Here, m is the number of observations in the test set, and each term in the sum is calculated in the same manner as the training error rate, but using the test data. This metric gives an unbiased estimate of the model's performance on new, unseen data.

5.1 Linear regression with OLS as supervised learning

5.1.1 Basic linear regression and OLS approach

Supervised learning is essentially about estimating a function f based on a set of input-output pairs, where $Y = f(X) + \epsilon$.

EvalOnce an estimate $\hat{f}$ of f is developed using different algorithms or methods (parametric or non-parametric), it's crucial to evaluate how well $\hat{f}$ predicts new data. This evaluation is often based on the mean squared error (MSE), decomposed into bias and variance components to understand the trade-offs involved.

Definition 5.1: General linear regression model

Model Formulation: In linear regression, the outcome Y is assumed to be a linear combination of the input features X, plus an error term ϵ. The equation is expressed as:

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon$$

Here, $\beta_0, \beta_1, \ldots, \beta_p$ are coefficients that quantify the influence of each feature on the outcome. β_0 is known as the intercept.

Assumptions of linear model

Assume $(y_i, \mathbf{x}_i)$, for $1 \leq i \leq n$, are independent and satisfy

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \epsilon_i$$

where

- **Uncorrelatedness:** ϵ_j is uncorrelated with ϵ_i
- **Linearity:** $\mathbb{E}[\epsilon_i] = 0$
- **Homoscedasticity:** $\mathrm{Var}(\epsilon_i) = \sigma^2$
- **Normality:** $\epsilon_i \sim N(0, \sigma^2)$

Given estimates $\hat{\beta}_j$ of the coefficients β_j, the prediction for a new input $X = x$ is made using:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p$$

One of the way to estimate the coefficients is using the ordinary least square approach. Recall that we want to find a function g by minimizing MSE

$$\min_g \mathbb{E}\left[(Y - g(X))^2\right]$$

Under the given linear model, it means we need to find $\alpha_0, \ldots, \alpha_p$ by performing:

$$\min_{\alpha_0,\ldots,\alpha_p} \mathbb{E}\left[(Y - \alpha_0 - \alpha_1 X_1 - \cdots - \alpha_p X_p)^2\right]$$

Just as we introduced above, in the model fitting step, we use the training data to approximate the above expectation (w.r.t. X and Y). Specifically, given $\mathcal{D}^{\text{train}}$, we choose

$$\left(\hat{\beta}_0, \cdots, \hat{\beta}_p\right) = \underset{\alpha_0,\cdots,\alpha_p}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \alpha_0 - \alpha_1 x_{i1} - \cdots - \alpha_p x_{ip}\right)^2 .$$

To simplify computation, particularly when dealing with multiple regression where there are several predictors, the OLS problem is often expressed and solved in matrix notation.

- $\mathbf{X}$: a $n \times (p+1)$ matrix of feature values, where each row represents an instance and includes a leading 1 for the intercept term, α_0.
- $\boldsymbol{\beta}$: a vector of size $(p+1) \times 1$ containing the true coefficients.
- $\boldsymbol{\alpha}$: a vector of size $(p+1) \times 1$ representing the coefficients to be estimated.
- $\boldsymbol{y}$: a vector of size $n \times 1$ containing the observed outcomes.

The OLS estimator can then be written as:

$$\hat{\boldsymbol{\beta}} = \underset{\alpha \in \mathbb{R}^{p+1}}{\operatorname{argmin}} \frac{1}{n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\alpha}\|_2^2$$

The solution to this optimization problem involves setting the derivative of the cost function (the sum of squared residuals) with respect to $\boldsymbol{\alpha}$ to zero, leading to the normal equations:

$$\mathbf{X}^T \mathbf{X} \boldsymbol{\alpha} = \mathbf{X}^T \mathbf{y}$$

Solving these equations gives:

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y}$$

This solution assumes that $\mathbf{X}^T\mathbf{X}$ is non-singular (i.e., has full rank), meaning that the columns of $\mathbf{X}$ are linearly independent.

Properties and equations of $\hat{\beta}$

- Unbiasedness: $\mathbb{E}[\hat{\boldsymbol{\beta}}] = \boldsymbol{\beta}$
- Covariance of $\hat{\boldsymbol{\beta}}$: $\operatorname{Cov}(\hat{\boldsymbol{\beta}}) = \sigma^2 \left(\mathbf{X}^\top \mathbf{X}\right)^{-1}$
- Mean Squared Error (MSE) of $\hat{\beta}$: $\mathbb{E}\left[\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\|_2^2\right] = \sigma^2 \operatorname{trace}\left[\left(\mathbf{X}^\top \mathbf{X}\right)^{-1}\right]$
- Special Case: $\mathbf{X}^\top \mathbf{X} = n\mathbf{I}_{p+1}$, the MSE becomes $\mathbb{E}\left[\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\|_2^2\right] = \frac{\sigma^2(p+1)}{n}$. So we have a dependence on p and sample size n.

Remark.

The general form for minimizing the training error in supervised learning in general is:

$$\hat{f} = \underset{g \in \mathcal{F}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, g\left(\mathbf{x}_i\right)\right)$$

- $\mathcal{F}$: This is the set of all possible functions (or models) that could be used to predict Y from X. For linear regression, $\mathcal{F}$ consists of linear functions of the form $\mathbf{x}^\top \boldsymbol{\beta}$.
- $L\left(y_i, g\left(\mathbf{x}_i\right)\right)$: This is the loss function that measures the difference between the true output y_i and the predicted output $g\left(\mathbf{x}_i\right)$. Different supervised learning methods use different loss functions depending on

the problem (regression or classification).

- $\frac{1}{n}\sum_{i=1}^{n}$: This represents the average loss over all n training data points. The goal is to minimize this average loss to find the best possible predictor $\hat{f}$.

Remark.

OLS can be seen as a specific case of this general framework. In OLS, the loss function $L(a, b)$ is the squared difference between the actual and predicted values:

$$L(a, b) = (a - b)^2$$

Remark.

The class of functions $\mathcal{F}(x)$ in OLS is the set of all linear functions of x :

$$\mathcal{F}(x) = \left\{ \mathbf{x}^\top \boldsymbol{\beta} : \boldsymbol{\beta} \in \mathbb{R}^{p+1} \right\}$$

Remark.

In terms of linear regression, we cannot hope $\hat{\beta}_0 = \beta_0$ and $\hat{\beta}_1 = \beta_1$, because they depend on the observed data which is random. The fitted OLS lines are thus also different, but their average is close to the true regression line, i.e.

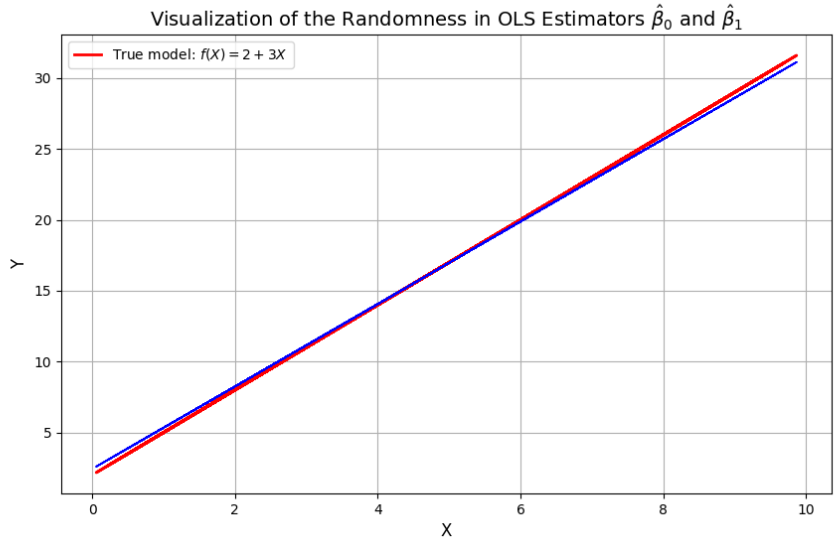

Figure 3: True linear model vs. predicted model

5.1.2 Inference on the coefficients: Estimating variance

In linear regression, σ^2 represents the variance of the error terms ϵ, which capture the difference between the observed values y_i and the predicted values $\mathbf{x}_i^\top \hat{\boldsymbol{\beta}}$. However, σ^2 is unknown and needs to be estimated from the data. The estimated variance $\hat{\sigma}^2$ is given by:

$$\hat{\sigma}^2 = \frac{1}{n - p - 1} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^\top \hat{\boldsymbol{\beta}} \right)^2$$

- n is the number of observations (sample size).
- p is the number of predictors (excluding the intercept term).
- $\hat{\boldsymbol{\beta}}$ represents the estimated coefficients.

- $\mathbf{x}_i^\top \hat{\boldsymbol{\beta}}$ is the predicted value for the i-th observation.

5.1.3 Inference on the coefficients: confidence interval on coefficients

A confidence interval gives a range of plausible values for a coefficient β_j, based on the estimated value $\hat{\beta}_j$ and its standard error. In this case, we are focusing on a 95% asymptotic confidence interval for each coefficient β_j, which has the form:

$$\left[\hat{\beta}_j - 1.96 \cdot SE\left(\hat{\beta}_j\right), \hat{\beta}_j + 1.96 \cdot SE\left(\hat{\beta}_j\right)\right]$$

- $\hat{\beta}_j$ is the estimated coefficient for the j-th predictor.
- $SE\left(\hat{\beta}_j\right)$ is the standard error of the estimated coefficient, which measures the uncertainty around the estimate.

The standard error $SE\left(\hat{\beta}_j\right)$ is calculated as:

$$SE\left(\hat{\beta}_j\right) = \sqrt{\hat{\sigma}^2 \left[(\mathbf{X}^\top \mathbf{X})^{-1}\right]_{jj}}$$

- $\hat{\sigma}^2$ is the estimated variance.
- $\left[\left(\mathbf{X}^\top \mathbf{X}\right)^{-1}\right]_{jj}$ is the j-th diagonal element of the matrix $\left(\mathbf{X}^\top \mathbf{X}\right)^{-1}$, which reflects how much uncertainty there is in the estimate of β_j.

Remark.

> An asymptotic confidence interval refers to a confidence interval that is valid in large samples. It is based on the assumption that as the sample size grows to infinity. The reason is in small samples, the distribution of $\hat{\beta}_j$ might not be well approximated by a normal distribution, making the confidence interval less reliable. However, as the sample size increases, the approximation improves, and the confidence interval becomes more accurate.

5.1.4 Inference on the coefficients: hypothesis testing on coefficients

Hypothesis testing allows us to formally test whether there is a statistically significant relationship between a predictor X_j and the response Y. The Null and Alternative Hypotheses are:

- Null Hypothesis (H_0) : $\beta_j = 0$ (there is no linear relationship between X_j and Y).
- Alternative Hypothesis (H_1) : $\beta_j \neq 0$ (there is a linear relationship between X_j and Y).

In this case, you are testing whether the predictor X_j has a significant impact on Y.

To test the hypothesis, we calculate a t-statistic:

$$t = \frac{\hat{\beta}_j}{SE\left(\hat{\beta}_j\right)}$$

Under the null hypothesis $(H_0 : \beta_j = 0)$, the test statistic follows a t -distribution with $n - p - 1$ degrees of freedom. The p-value represents the probability of observing a value of t as extreme as the one calculated, under the assumption that H_0 is true. If the p-value is less than or equal to 0.05 , we typically reject the null hypothesis, indicating that there is a significant linear relationship between X_j and Y.

The hypothesis testing procedure can be generalized to test whether a group of coefficients are all equal to zero. For example, you might want to test:

$$H_0 : \beta_{p-q+1} = \beta_{p-q+2} = \cdots = \beta_p = 0$$

This tests whether a subset of predictors (from X_{p-q+1} to X_p) have any significant relationship with Y. To conduct this test, we use an F-test or partial F-test, which is a generalization of the t-test for multiple coefficients.

5.1.5 Coefficient of determination

Definition 5.2: Residuals and RSS

For each data point $(\mathbf{x}_i, y_i)$, the fitted value $\hat{y}_i$ is calculated using the regression model:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}$$

The difference between the observed value y_i and the fitted value $\hat{y}_i$ is called the residual:

$$\text{Residual } = y_i - \hat{y}_i$$

The Residual Sum of Squares (RSS) is the sum of the squared residuals across all data points:

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

This is equivalent to the training Mean Squared Error (MSE) of the model, except without dividing by the number of observations n. The smaller the RSS, the better the model fits the data.

Definition 5.3: TSS

The Total Sum of Squares (TSS) measures the total variation in the observed Y values. It is defined as:

$$TSS = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

where $\bar{y}$ is the mean of the observed Y values:

$$\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$$

The TSS represents the total variance in Y, or how much Y varies around its mean. It quantifies the overall variability of the outcome Y in the sample.

Definition 5.4: R square

R^2 measures the proportion of the variation in the outcome Y that can be explained by the predictors X. The R^2 value is calculated as:

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}$$

This formula shows that R^2 is the proportion of the total variability in Y (TSS) that is explained by the model (through the fitted values). Here's how to interpret it:

- $R^2 = 1$: The model explains all the variability in Y (perfect fit).
- $R^2 = 0$: The model explains none of the variability in Y (no relationship between X and Y).

Remark.

A high R^2 does not necessarily mean that the model fits the data well. R^2 tends to increase with the number of predictors, even if those predictors do not meaningfully improve the model. This means that more complex models (with more predictors or more flexible structures) will often have a higher R^2, but they may be overfitting the data.

Remark.

To counteract this problem, Adjusted R^2 can be used (which adjusts for the number of predictors). This will be discussed in later sections of the PPT.

5.2 Model selection

5.2.1 Golden rule for best model

Given several candidate models, how do we choose the one that is most appropriate for the data?

Example.

- **Model 1:** $Y = \alpha_0 + \alpha_1 X_1 + \epsilon$
- **Model 2:** $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$

These two models will lead to different predictions for new data points $X = \mathbf{x} = (x_1, x_2)$:

$$\hat{f}_1(x) = \hat{\alpha}_0 + \hat{\alpha}_1 x_1 \quad \text{vs} \quad \hat{f}_2(x) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$$

The goal is to identify and choose the model that has the smaller expected Mean Squared Error (MSE). When you have access to a test set $\mathcal{D}_{\text{test}}$, you can compare the test MSEs directly:

$$\frac{1}{m}\sum_{i=1}^{m}\left(y_i^{(T)} - \hat{\alpha}_0 - \hat{\alpha}_1 x_{i1}^{(T)}\right)^2 \quad \text{vs.} \quad \frac{1}{m}\sum_{i=1}^{m}\left(y_i^{(T)} - \hat{\beta}_0 - \hat{\beta}_1 x_{i1}^{(T)} - \hat{\beta}_2 x_{i2}^{(T)}\right)^2$$

We would choose the model with the smaller test MSE because it generalizes better to new, unseen data.

5.2.2 When there is no testing set

When no test set $\mathcal{D}_{\text{test}}$ is available, we need alternative methods to estimate the expected MSE. There are two main approaches to model selection when a test set isn't available:

1. **Direct estimating the MSE:** we can directly estimate the expected MSE by creating a "test set" from the training data itself using data-splitting techniques:
   - *Validation Set Approach:* Split the data once into a training set and a validation set. Train the model on the training set and evaluate the performance on the validation set.
   - *Cross-Validation:* Split the data multiple times and average the results. This is a more robust method as it reduces variability in the estimation of the test error.
2. **Model complexity adjustment:** Rather than creating a test set, we adjust the training error to account for model complexity. This approach is done using criteria such as:
   - *Mallow's C_p :* Adjusts for the number of predictors in the model.
   - *Adjusted R^2 :* Adjusts the R^2 statistic to penalize the use of too many predictors.
   - *Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC):* Both penalize model complexity by adding terms that account for the number of parameters in the model.

5.2.3 Direct estimation of expected MSE: Validation set approaches

The validation set approach is a straightforward method for estimating the expected test error. The data is randomly divided into two parts: a training set (used to fit the model) and a validation set (used to evaluate the model's performance). The exact proportion used for splitting depends on the size of the dataset, but common ratios are 70% for training and 30% for validation.

The model is trained using the training set, and the validation set is used to compute the Mean Squared Error (MSE). The resulting validation-set MSE provides an estimate of the model's expected MSE, giving a sense of how well the model generalizes to unseen data.

Example.

Consider the dataset with 392 observations is split into two equal sets (196 observations for training and 196 for validation). The left-hand side of the plot shows the validation error estimates from a single random split. The right-hand side shows the results of repeating this process 10 times with different random splits.

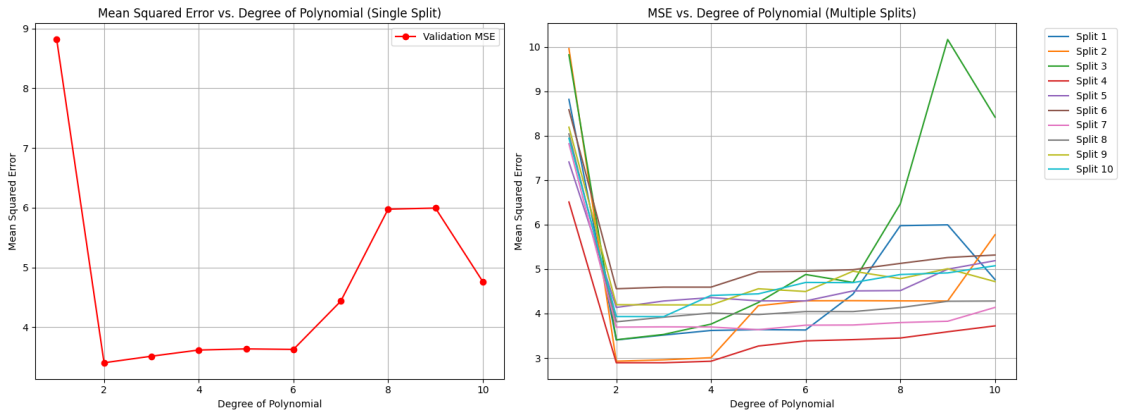

Figure 4: Single split and multiple splits yield different results.

For a fixed polynomial degree, each time you randomly split the dataset into a training set and a validation set, the composition of those sets changes. Since the observations in the training and validation sets vary with each random split, the model fitted on the training set and the resulting validation MSE can also change significantly. This randomness leads to instability in the validation set approach.
Also, since only a subset of the data (the training set) is used to fit the model, the model may be underfitting the data. This means the validation set error might overestimate the actual test error that would occur if the model were trained on the entire dataset.

Thus, we conclude that the validation set approach is in fact unstable and might lead to underfitting. This means we need different approaches to estimate for expected MSE.

5.2.4 Direct estimation of expected MSE: LOOCV as cross-validation

To address the instability of the validation set approach, Leave-One-Out Cross-Validation (LOOCV) is introduced as a more robust method.

LOOCV process

1. For each observation i in the dataset, we treat it as a validation set and use the remaining $n-1$ observations to form the training set.
2. The model is trained on the $n-1$ observations, and we predict the value of the left-out observation $\mathbf{x}_i$.

For each iteration, the test error is calculated as:

$$\mathrm{MSE}_i = \left(y_i - \hat{f}_i\left(\mathbf{x}_i\right)\right)^2$$

This process is repeated for all n observations, leaving each one out in turn. The LOOCV estimate for the test MSE is the average of the individual test errors:

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i$$

Unlike the validation set approach, LOOCV is more stable because the training set in each iteration is almost the entire dataset. The results are consistent, as there is no randomness in the splitting.

Computational expense is the major drawback of LOOCV. Since the model needs to be fitted n times, it can be computationally expensive, especially for large datasets.

5.2.5 Direct estimation of expected MSE: k-Fold as cross-validation

To balance the computational cost of LOOCV and the instability of the validation set approach, k-fold cross-validation is introduced.

Process of K-fold

1. The data is randomly divided into k roughly equal-sized folds.
2. The model is trained on $k-1$ folds, and the remaining fold is used as the validation set.
3. This process is repeated k times, with each fold serving as the validation set once.

For each fold i, the Mean Squared Error (MSE) is computed for the observations in that fold. The k -fold CV estimate of the test error is the average of these k errors:

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i$$

LOOCV is essentially n-fold cross-validation, where each fold contains a single observation. In practice, 5-fold or 10-fold cross-validation is widely used. These values balance computational efficiency and stability.

5.2.6 Direct estimation of expected MSE: cross-validation for special classification problem

Cross-validation methods can also be applied to classification problems, where the response variable Y is categorical. LOOCV for classification: The dataset is split in the same way as for regression. For each observation i, we compute the classification error on the validation set:

$$\mathrm{Err}_i = 1\left\{y_i \neq \hat{f}\left(\mathbf{x}_i\right)\right\}$$

Here, Err_i is 1 if the model incorrectly classifies observation i and 0 otherwise. The LOOCV estimate for classification is the average classification error across all n observations:

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{Err}_i$$

5.2.7 Problems with cross-validations

The primary goal of cross-validation is to evaluate a model's performance on unseen data by simulating the test set with different portions of the available data. You want to ensure that the model is trained on part of the data (training set) and evaluated on a separate, independent part (validation set).

The core principle behind cross-validation is that the validation set must be independent from the training process. That means the model should not "see" or use any information from the validation set during its fitting or training.

If this independence is violated, the validation error (test error) estimate will be biased. It may look like the model performs better than it actually does because the model has, in some way, already seen or used the validation set.

Example.

> You start with 5000 predictors (features) and 100 samples (data points). Since having 5000 predictors is impractical for fitting a model, you use a preliminary method to identify 10 predictors that have the largest correlation with the outcome variable Y. After selecting the 10 best predictors from step 1, you apply Ordinary Least Squares (OLS) to these 10 predictors to fit a linear regression model.
> Suppose you use cross-validation to estimate the model's test error, the issue arises because the 10 predictors you selected were based on the entire dataset, including what would eventually be used as the validation set. What happens? The validation set was indirectly used to identify the 10 predictors, which means the predictors were chosen with information from the entire dataset. This causes the independence between the validation set and the model fitting process to break down.
> Since the model's selection process (choosing the 10 predictors) was influenced by the validation set, the cross-validation error estimate will likely underestimate the true test error. This makes it look like the model performs better than it really does, giving you a false sense of confidence in the model.

5.2.8 Avoid estimating the expected MSE: C_p, AIC, BIC, and adjusted R^2

For these methods, they offer a way to select models by adjusting for the complexity of the model. These methods are limited to:

1. Parametric models (e.g., linear models).
2. Models where the likelihood of the data is correctly specified, meaning we assume a specific distribution for the error terms (often Gaussian).

Definition 5.5: Mallow

Mallow's C_p is a model selection criterion that adjusts for the number of predictors in the model. It adds a penalty to the training MSE to account for model complexity:

$$C_p(\hat{f}) = \frac{1}{n} RSS(\hat{f}) + \frac{2p\sigma^2}{n}$$

- p is the number of parameters in the model.
- σ^2 is the error variance, which can be estimated by $\hat{\sigma}^2$ if unknown.

The penalty term $\frac{2p\sigma^2}{n}$ compensates for the model's complexity and helps prevent overfitting. The goal is to select the model with the **lowest C_p value**.

Definition 5.6: Adjusted R square

The Adjusted R^2 statistic is a modification of the regular R^2 statistic, which adjusts for the number of predictors in the model. Recall that the total sum of squares (TSS) is:

$$TSS = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

The regular R^2 is defined as:

$$R^2(\hat{f}) = 1 - \frac{RSS(\hat{f})}{TSS}$$

However, the Adjusted R^2 accounts for the number of predictors p and the sample size n :

$$\text{Adjusted } R^2(\hat{f}) = 1 - \frac{RSS(\hat{f})/(n-p-1)}{\text{TSS}/(n-1)}$$

Remark.

Unlike R^2, Adjusted R^2 penalizes models for including unnecessary predictors. A larger value of Adjusted R^2 suggests a model with smaller test error. While RSS always decreases as p increases, the term $\frac{RSS}{n-p-1}$ may increase or decrease, reflecting the impact of model complexity.

Remark.

Both Mallow's C_p and Adjusted R^2 are restricted to the selection of linear models.

Definition 5.7: AIC

AIC is a model selection criterion that applies to models derived from the maximum likelihood estimation (MLE) approach. It penalizes models based on their complexity by adding a term proportional to the number of parameters p. The AIC is defined as:

$$AIC(\hat{f}) = -2\log L(\hat{f}) + 2p$$

Where:

- $L(\hat{f})$ is the maximum value of the likelihood function.
- The term $2p$ is the penalty for the number of parameters in the model.

Theorem 5.8: AIC and Mallow equivalence

For linear models where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, AIC is proportional to Mallow's C_p and will select the same model.

Proof. For a linear model with normal noise, we have:

$$y_i = x_i^T\beta + \varepsilon_i \quad \text{where} \quad \varepsilon_i \sim \mathcal{N}\left(0, \sigma^2\right) \quad \text{i.i.d. for} \quad i = 1, \ldots, n$$

And the loss function is given by:

$$L(\beta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(y_i - x_i^T\beta\right)^2}{2\sigma^2}\right)$$

The log-likelyhood function becomes:

$$\begin{aligned}\log L(\beta) &= \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(y_i - x_i^T\beta\right)^2}{2\sigma^2}\right)\right) \\ &= -\frac{n}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y_i - x_i^T\beta\right)^2\end{aligned}$$

We know AIC is:

$$\mathrm{AIC}(\beta) = 2k - 2\log L(\beta)$$

Where k is the number of parameters:

$$\begin{aligned}\mathrm{AIC}(\beta) &= 2p - 2\left(-\frac{n}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y_i - x_i^T\beta\right)^2\right) \\ &= 2p + n\log\left(2\pi\sigma^2\right) + \frac{1}{\sigma^2}\sum_{i=1}^{n}\left(y_i - x_i^T\beta\right)^2\end{aligned}$$

Dropping constant terms (as they do not affect minimization):

$$\mathrm{AIC}(\beta) \approx \frac{1}{\sigma^2}\mathrm{RSS}(\beta) + 2p$$

We have C_p Statistic:

$$C_p(\beta) = \frac{1}{n}\mathrm{RSS}(\beta) + \frac{2p\sigma^2}{n}$$

Multiplying $C_p(\beta)$ by $\frac{n}{\sigma^2}$ gives:

$$\frac{n}{\sigma^2}C_p(\beta) = \frac{1}{\sigma^2}\mathrm{RSS}(\beta) + 2p$$

This is exactly the same formula as AIC modulo a constant factor. The AIC and C_p are proportional when the variance σ^2 is estimated consistently, meaning that minimizing the AIC and minimizing the C_p will lead to the same choice of model parameters β for a linear model with Gaussian errors. ■

Remark.

- We select the model with the lowest AIC value.

Definition 5.9: BIC

BIC is similar to AIC but imposes a stronger penalty for models with many parameters. The BIC is given by:

$$BIC(\hat{f}) = -2 \log L(\hat{f}) + (\log n)p$$

Where:

- $\log n$ is the logarithm of the sample size.
- p is the number of parameters.

Remark.

BIC applies a heavier penalty than AIC because the penalty term grows with $\log n$. As a result, BIC tends to select simpler models than AIC. Like AIC, BIC is used for parametric models and selects the model with the lowest BIC value.

5.2.9 C_p, AIC, BIC, and adjusted R^2: comparison and applications

Let's compare the different methods:

1. Mallow's C_p : Adds a penalty based on the number of parameters p. Mainly used for linear models.
2. Adjusted R^2 : Adjusts the R^2 statistic to penalize for unnecessary predictors.
3. AIC: Penalizes the likelihood of the model based on the number of parameters, with a smaller penalty than BIC.
4. BIC: Similar to AIC but with a stronger penalty for complex models, favoring simpler models.

Example.

Two models are compared:

- Model 1: $Y = \alpha_0 + \alpha_1 X_1 + \epsilon$
- Model 2: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$

Their residual sum of squares (RSS) can be computed as:

$$\text{RSS}\left(\hat{f}_1\right) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\alpha}_0 - \hat{\alpha}_1 x_{i1})^2$$

$$\text{RSS}\left(\hat{f}_2\right) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2}\right)^2$$

Mallow's C_p

$$C_p\left(\hat{f}_1\right) = \frac{1}{n}\left(RSS\left(\hat{f}_1\right) + (2 \times 1)\sigma^2\right)$$

$$C_p\left(\hat{f}_2\right) = \frac{1}{n}\left(RSS\left(\hat{f}_2\right) + (2 \times 2)\sigma^2\right)$$

Adjusted R^2

$$\text{Adjusted } R^2\left(\hat{f}_1\right) = 1 - \frac{\text{RSS}\left(\hat{f}_1\right)/(n-2)}{\text{TSS}/(n-1)}$$

$$\text{Adjusted } R^2\left(\hat{f}_2\right) = 1 - \frac{RSS\left(\hat{f}_2\right)/(n-3)}{TSS/(n-1)}$$

5.2.10 Model selection approaches: data splitting vs. criteria based techniques

Data-Splitting Techniques (e.g., cross-validation):

1. **Advantages:**
   - Provides a direct estimate of the test error.
   - Can be used for a wide variety of models, including non-parametric models where it's difficult to estimate degrees of freedom or variance.
2. **Drawbacks:**
   - Requires a large sample size to be effective.
   - Difficult to guarantee that the model selected via cross-validation is optimal.

Criteria-Based Techniques (e.g., C_p, AIC, BIC, Adjusted R^2):

1. **Advantages:**
   - Better suited for smaller datasets.
   - When the distribution is well-specified and the variance is known or estimable, these methods are preferred.
2. **Drawbacks:** Limited to parametric models.

5.3 Improving the OLS estimator for linear regression

5.3.1 Limitation of OLS

Although OLS is a powerful tool, it has limitations, especially when dealing with high-dimensional data or noisy datasets. Two main issues with the OLS estimator are:

- **Large variance when p is large:** When the number of predictors p is large relative to the sample size n, the OLS estimator can have high variance, making the model overly sensitive to small fluctuations in the data. If $p > n$ (more predictors than observations), the OLS estimator is not unique and its variance becomes infinite, making the model unreliable.
- **Lack of interpretability:** Including irrelevant features in the model can make it difficult to interpret. By setting some coefficient estimates to zero (effectively removing unnecessary features), we can achieve a more parsimonious and interpretable model.

5.3.2 Ways to improve the OLS

To address the limitations of OLS, we can explore several approaches:

1. **Subset selection:** In this method, we identify a subset of the predictors $X_1, X_2, \ldots, X_p$ that we believe are related to the response Y. We then apply OLS to this subset, excluding irrelevant predictors. This results in a more interpretable and possibly more accurate model.
2. **Shrinkage (Regularization):** Instead of eliminating predictors, we include all the predictors but shrink the estimated coefficients towards zero. This approach, known as regularization, helps reduce variance. Shrinkage methods such as Ridge regression and Lasso regression not only reduce the variance but can also perform variable selection by setting some coefficients exactly to zero (in the case of Lasso).
3. **Dimension reduction:** This technique projects the original p predictors into an $\mathbf{M}$-dimensional subspace where $M < p$. Dimension reduction methods, such as Principal Component Regression (PCR), create linear combinations of the original predictors and then apply OLS to the reduced set of predictors.

5.3.3 Subset selection

Definition 5.10: Best subset selection

Best Subset Selection involves evaluating all possible subsets of the predictors and selecting the subset that provides the best model fit. This method ensures that the best possible model is selected, but it can be computationally expensive.

Algorithm for best subset selection

1. Start with the null model $\mathcal{M}_0$, which contains no predictors and simply predicts the mean of the response.
2. For $k = 1, 2, \ldots, p$:
   - Fit all possible models that contain exactly k predictors (i.e., all $\binom{p}{k}$ models).
   - Select the model with the smallest RSS or the highest R^2, and call it $\mathcal{M}_k$.
3. Select the best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using a model selection criterion such as cross-validated prediction error, C_p (AIC), BIC, or Adjusted R^2.

Remark.

In Step 2, we use RSS because it is a natural and consistent measure of the model's fit to the data, with smaller RSS indicating a better fit. R^2 can also be used.

Remark.

For step 3, we do not use RSS or R^2 since they will always favor the complex number, since RSS monotonically decreases as the model becomes complex, and R^2 monotonically increases as model becomes complex.

Remark.

Best subset selection involves fitting and comparing $\binom{p}{0} + \binom{p}{1} + \binom{p}{2} + \cdots + \binom{p}{p} = 2^p$ models. This becomes computationally infeasible when p is large.

Definition 5.11: Forward stepwise selection

Forward Stepwise Selection is a computationally more efficient alternative to Best Subset Selection. Instead of evaluating all possible subsets of predictors, it builds the model step by step by adding one predictor at a time.

Algorithm for forward stepwise selection

1. Start with the null model $\mathcal{M}_0$, which contains no predictors.
2. For $k = 0, \ldots, p-1$:
   - Consider all models that add one predictor to the current set of k predictors.
   - Select the best model (the one with the smallest RSS or largest R^2) and call it $\mathcal{M}_{k+1}$.
3. Select the best model from $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validation, C_p AIC, BIC, or adjusted R^2.

Remark.

Forward Stepwise Selection is computationally faster than Best Subset Selection because, in each iteration, it only needs to evaluate $p - k$ models, rather than all possible subsets. The total number of models considered is $1 + \sum_{k=0}^{p-1}(p-k) = 1 + \frac{p(p+1)}{2}$, which is significantly fewer than 2^p models.

Remark.

Since it adds predictors one at a time, it may miss the best possible model that could have been found by evaluating all subsets. It's not guaranteed to find the global optimum model from among the 2^p possible subsets.

Definition 5.12: Backward stepwise selection

Backward Stepwise Selection works in the opposite direction of Forward Stepwise Selection. It starts with the full model, which includes all predictors, and then removes one predictor at a time.

Algorithm for backwards stepwise selection

1. Start with the full model $\mathcal{M}_p$, which includes all p predictors.
2. For $k = p, p-1, \ldots, 1$:
   - Consider all models that remove one predictor from the current set of k predictors.
   - Select the best model (smallest RSS or largest R^2) and call it $\mathcal{M}_{k-1}$.
3. Select the best model from $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validation, C_p, AIC, BIC, or adjusted R^2.

Remark.

Like Forward Stepwise Selection, it evaluates far fewer models than Best Subset Selection, as it only evaluates $1 + \frac{p(p+1)}{2}$ models.

Remark.

Backward Selection is useful when p is small, but Forward Selection is preferred when $p > n$ because Backward Selection requires the full model to be fitted first, which is not feasible when $p > n$.

Remark.

It is also a greedy procedure, meaning it is not guaranteed to find the best model, as it may remove a predictor that would have worked well when combined with others.

5.3.4 Shrinkage regression

Shrinkage Methods involve fitting a model using all p predictors, but with an added penalty that shrinks the coefficient estimates towards zero. This shrinking reduces the variance of the coefficient estimates, which can significantly improve model performance by reducing overfitting.

The two most well-known shrinkage methods are ridge regression and LASSO regression.

Definition 5.13: Ridge regression

The Ordinary Least Squares (OLS) estimator estimates the regression coefficients $\beta_0, \ldots, \beta_p$ by minimizing the residual sum of squares (RSS):

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

In Ridge Regression, an additional shrinkage penalty is applied to the coefficients:

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

$\lambda \geq 0$ is a tuning parameter (also called the regularization parameter) that controls the amount of shrinkage. When $\lambda = 0$, Ridge Regression reduces to OLS.

Remark.

The Ridge estimator is denoted as $\hat{\boldsymbol{\beta}}_{\lambda}^{R} = \operatorname{argmin}_{\beta=(\beta_0,\ldots,\beta_p)\in\mathbb{R}^{p+1}} \underbrace{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2}_{RSS} + \lambda\sum_{j=1}^{p}\beta_j^2$

Remark.

The second term $\lambda\sum_{j=1}^{p}\beta_j^2$ is the penalty that shrinks the coefficient estimates β_j towards zero.

Remark.

In Ridge Regression, the penalty term forces the model to sacrifice a perfect fit to the training data (slightly increasing the RSS) in exchange for smaller, more stable coefficient estimates. This means that the RSS will be slightly larger compared to OLS because the model is not allowed to fit the data as closely due to the shrinking of the coefficients. Thus, we have a trade-off between bias and variance.

Remark.

A good choice of λ is critical. If λ is too small, Ridge Regression behaves like OLS, while if λ is too large, the coefficients shrink too much, and the model underfits the data. Cross-validation is often used to find the optimal value of λ.

Remark.

The overall test MSE is able to be minizmied at optimal λ.

Remark.

In Ridge Regression, it is important to standardize the predictors, i.e., scale them so that each has a standard deviation of 1 . This prevents the penalty from affecting predictors with large scales more than others. The standardization formula is:

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}}$$

The ridge regression has the following advantages:

1. **Better Prediction:** Ridge Regression often provides better prediction than OLS by shrinking the coefficient estimates and trading off bias for reduced variance, Ridge Regression models are more stable and less sensitive to the training data.
2. **Computational Efficiency:** Ridge Regression is computationally efficient and is much faster than methods like best subset selection.

The ridge regression also has limitations. One major limitation of Ridge Regression is that it does not perform variable selection. That is, it does not set any coefficient estimates to exactly zero. As a result, Ridge Regression typically includes all p predictors in the model, making it difficult to interpret the results and determine which predictors are most important.

Definition 5.14: LASSO regression

The Lasso (Least Absolute Shrinkage and Selection Operator) shrinks coefficients by penalizing their absolute values. The goal is to minimize the following objective function:

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

Remark.

Ridge Regression shrinks the coefficients using the ℓ_2-norm:

$$\|\boldsymbol{\beta}\|_2^2 = \sum_{j=1}^{p} \beta_j^2$$

Lasso uses the ℓ_1-norm:

$$\|\boldsymbol{\beta}\|_1 = \sum_{j=1}^{p} |\beta_j|$$

Remark.

Like Ridge Regression, the Lasso shrinks the coefficient estimates towards zero. However, the ℓ_1 -penalty in Lasso forces some coefficients to be exactly zero when λ is large enough. Therefore, Lasso not only shrinks coefficients but also performs variable selection, effectively eliminating irrelevant predictors.

Remark.

A model with many coefficients equal to zero is called a sparse model. Sparsity is desirable because it leads to simpler, more interpretable models.

Remark.

Similar to Ridge Regression, selecting the optimal value of λ is crucial. Cross-validation is typically used to find the best λ.

An alternative formulation of the Lasso and Ridge Regression problems is to minimize the RSS subject to a constraint on the size of the coefficients:

- Lasso minimizes the RSS, subject to a constraint on the ℓ_1-norm of the coefficients:

$$\underset{\beta}{\text{minimize}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \leq s$$

- Ridge Regression minimizes the RSS, subject to a constraint on the ℓ_2-norm of the coefficients:

$$\underset{\beta}{\text{minimize}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 \leq s$$

Here, $s \geq 0$ is a regularization parameter that controls the strength of the constraint (similar to λ).

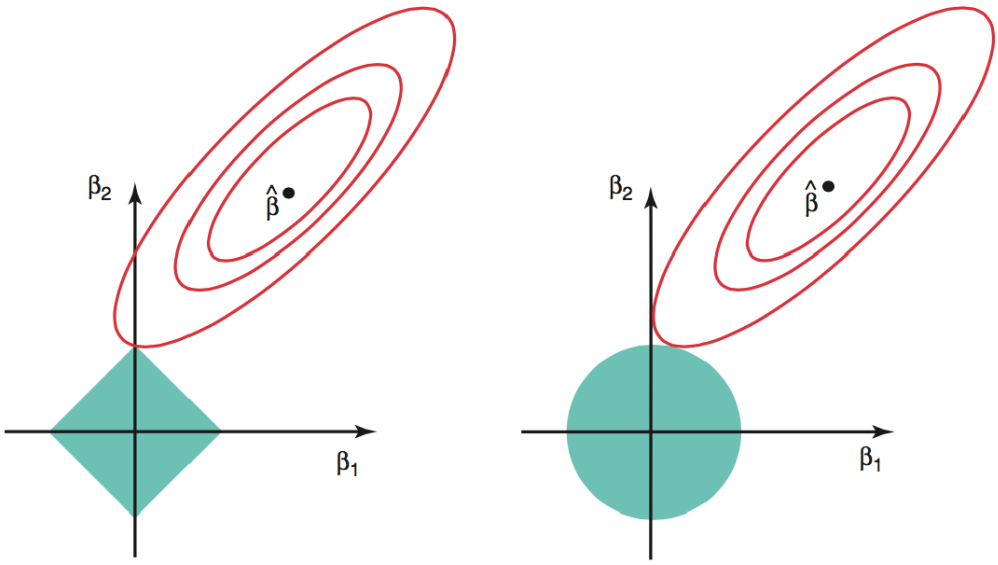

Figure 5: Lasso yields zero estimates

The red ellipses represent contours of the RSS (Residual Sum of Squares). Each successive ellipse outward represents a higher level of RSS. The center (the intersection of the axes where the ellipses are tightest) is the point where the RSS is minimized with no regularization constraint applied (i.e., the OLS estimate). The goal is to optimize until the lowest possible RSS contour touches the constraint area.

1. For the left hand side. The solid areas are the constraint regions, $|\beta_1| + |\beta_2| \leq s$ and $\beta_1^2 + \beta_2^2 \leq s$, while the red ellipses are the contours of the RSS.

   The corners of the diamond (where either β_1 or β_2 is zero) intersect the only possible RSS ellipse, indicating that Lasso can result in coefficients that are exactly zero. This sparsity effect occurs because the constraint region is most likely to touch the ellipse at its axes, which correspond to either β_1 or β_2 being zero.

2. For right hand side. The circular blue region depicts the Ridge constraint, formulated by $\beta_1^2 + \beta_2^2 \leq s$, representing an ℓ_2-norm.

   The circle intersects the ellipses in a way that generally does not align with the axes, implying that Ridge regression does not result in zero coefficients. Instead, it shrinks all coefficients toward zero evenly, but none of them reach zero exactly.

Remark.

When the true coefficients are non-sparse, Ridge Regression generally has lower variance than Lasso, leading to a smaller MSE. When the true coefficients are sparse, Lasso outperforms Ridge Regression by having both smaller bias and smaller variance.

Remark.

In general, there is no universal rule about whether Ridge or Lasso will perform better. It depends on the structure of the true data-generating process. But unlike Ridge Regression, Lasso performs variable selection, which is particularly useful for reducing the number of predictors and yielding more interpretable models.

Method to find optimal lamda

1. Create a grid of λ values to evaluate. This set can range over several orders of magnitude, typically on a logarithmic scale (e.g., $[0.001, 0.01, 0.1, 1, 10, 100]$).
2. Split the dataset into K roughly equal parts for K-fold cross-validation. Common choices for K are 5 or 10. Alternatively, use Leave-One-Out Cross-Validation (LOOCV) for small datasets.
3. For each value of λ, fit the model and compute the average of the validation set errors over all K folds to get the crossvalidation error for this λ.
4. Choose the λ that results in the lowest cross-validation error. This value of λ is considered optimal as it yields the best predictive performance by balancing bias (underfitting) and variance (overfitting).
5. Using the selected λ, refit the model on the entire dataset to finalize the model parameters.

Example.

In this simplified example, we will demonstrate the shrinkage effects of ridge and lasso. Assume $n = p$ and $\mathbf{X} = \mathbf{I}_n$, making each predictor X_j independent and directly associated with a corresponding response y_j without any interaction or correlation with other predictors.

$$\begin{bmatrix} y_1 \\ \vdots \\ y_p \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_p \end{bmatrix}$$

where ϵ_j are noise terms with zero mean and variance σ^2.

1. **For OLS estimators:**
   - *Objective:* Minimize $\sum_{j=1}^{p} (y_j - \beta_j)^2$.
   - *Solution:* $\hat{\beta}_j = y_j$ for all j, meaning each OLS estimate directly equals the corresponding response variable.
2. **For Ridge estimators:**
   - *Objective:* Minimize $\sum_{j=1}^{p} (y_j - \beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$, incorporating a penalty that shrinks coefficients toward zero.
   - *Solution:* $\hat{\beta}_j^R = \frac{y_j}{1+\lambda}$ for all j, showing how each estimate is scaled down, particularly as λ increases.
3. **LASSO estimators:**
   - *Objective:* Minimize $\sum_{j=1}^{p} (y_j - \beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$, favoring sparsity by potentially setting some coefficients exactly to zero.
   - *Solution:*

$$\hat{\beta}_j^L = \begin{cases} y_j - \frac{\lambda}{2} & \text{if } y_j > \frac{\lambda}{2} \\ y_j + \frac{\lambda}{2} & \text{if } y_j < -\frac{\lambda}{2} \\ 0 & \text{if } |y_j| \leq \frac{\lambda}{2} \end{cases}$$

   This is known as **soft-thresholding**, it adjusts coefficients by a constant amount and sets them to zero if they are within a specific range around zero.

Example.

Continue with the same example, now we look at the bias and variance of the OLS and ridge estimators.

1. **For OLS estimators:**

- *Bias:*
$$\mathbb{E}\left[\hat{\beta}_j\right] = \mathbb{E}\left[y_j\right] = \mathbb{E}\left[\beta_j + \epsilon_j\right] = \beta_j$$
- *Variance:*
$$\mathrm{Var}\left(\hat{\beta}_j\right) = \mathrm{Var}\left(\epsilon_j\right) = \sigma^2$$
- *Mean squared error of the j th coefficient:*
$$\mathbb{E}\left[\left(\hat{\beta}_j - \beta_j\right)^2\right] = \left(\mathbb{E}\left[\hat{\beta}_j\right] - \beta_j\right)^2 + \mathrm{Var}\left(\hat{\beta}_j\right) = \sigma^2$$
- *Mean squared error of all p coefficients:*
$$\mathbb{E}\left[\sum_{j=1}^{p}\left(\hat{\beta}_j - \beta_j\right)^2\right] = p\sigma^2$$

2. **For Ridge estimators:**

- *Bias:*
$$\mathbb{E}\left[\hat{\beta}_j^R\right] = \mathbb{E}\left[\frac{y_j}{1+\lambda}\right] = \mathbb{E}\left[\frac{\beta_j + \epsilon_j}{1+\lambda}\right] = \frac{\beta_j}{1+\lambda}$$
- *Variance:*
$$\mathrm{Var}\left(\hat{\beta}_j^R\right) = \mathrm{Var}\left(\frac{\epsilon_j}{1+\lambda}\right) = \frac{\sigma^2}{(1+\lambda)^2}$$
- *Mean squared error of the j th coefficient:*
$$\begin{aligned}\mathbb{E}\left[\left(\hat{\beta}_j^R - \beta_j\right)^2\right] &= \left(\mathbb{E}\left[\hat{\beta}_j^R\right] - \beta_j\right)^2 + \mathrm{Var}\left(\hat{\beta}_j^R\right) \\ &= \left(\frac{\beta_j}{1+\lambda} - \beta_j\right)^2 + \frac{\sigma^2}{(1+\lambda)^2} \\ &= \frac{\lambda^2\beta_j^2}{(1+\lambda)^2} + \frac{\sigma^2}{(1+\lambda)^2}\end{aligned}$$
- *Mean squared error of all p coefficients:*
$$\mathbb{E}\left[\sum_{j=1}^{p}\left(\hat{\beta}_j^R - \beta_j\right)^2\right] = \frac{\lambda^2\sum_{j=1}^{p}\beta_j^2 + p\sigma^2}{(1+\lambda)^2}$$

Definition 5.15: Elastic net

There are more options for penalties:

$$\underset{\boldsymbol{\beta}}{\mathrm{argmin}}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\left[(1-\alpha)\|\boldsymbol{\beta}\|_1 + \alpha\|\boldsymbol{\beta}\|_2\right]$$

for some tuning parameters $\lambda \geq 0$ and $\alpha \in [0,1]$. The ridge corresponds to $\alpha = 1$. The Lasso corresponds to $\alpha = 0$.

Definition 5.16: Group Lasso

We also have another form of penalties. If we suspect the model is nonlinear in X_1 or X_2, we can add quadratic terms, say

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_2 + \beta_4 X_2^2 + \epsilon$$

The group lasso estimator minimizes

$$RSS + \lambda \left(\sqrt{\beta_1^2 + \beta_2^2} + \sqrt{\beta_3^2 + \beta_4^2} \right)$$

In this penalty, we view β_1 and β_2 (coefficient of X_1 and X_1^2) as if they belong to the same group. The group Lasso can shrink the parameters in the same group (both β_1 and β_2) exactly to 0 simultaneously.

Definition 5.17: Extension to general penalties in non-linear models

The idea of penalization is generally applicable to almost all parametric models.

$$\hat{\boldsymbol{\beta}}_\lambda = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \underbrace{L\left(\boldsymbol{\beta}, \mathcal{D}^{\text{train}}\right)}_{g(\boldsymbol{\beta}; \mathcal{D}^{\text{train}})} + \lambda \cdot \operatorname{Pen}(\boldsymbol{\beta}).$$

- OLS: $L\left(\boldsymbol{\beta}, \mathcal{D}^{\text{train}}\right) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2, \operatorname{Pen}(\boldsymbol{\beta}) = 0.$
- Ridge: $L\left(\boldsymbol{\beta}, \mathcal{D}^{\text{train}}\right) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2, \operatorname{Pen}(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_2^2.$
- Lasso: $L\left(\boldsymbol{\beta}, \mathcal{D}^{\text{train}}\right) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2, \operatorname{Pen}(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_1.$

In principal, L can be any loss function and Pen could be any penalty function.

5.4 Moving beyond linearity in regression

To address limitations in the linear modeling of relationships, several extensions are considered:

1. **Univariate extensions $p = 1$ only one independent variable:**
   - *Polynomial Regression:* Expands the model to include higher-degree terms of a single predictor.
   - *Step Functions:* Breaks the range of x into intervals, fitting different constants in each.
   - *Regression Splines:* Uses piecewise polynomial functions joined at certain points called knots.
2. **Multivariate extensions $p > 1$ more than one independent variable:**
   - *Local Regression:* Fits a smooth curve through the data by giving more weight to nearby points.
   - *Generalized Additive Models (GAMs):* Allows the effect of each feature to be modeled nonparametrically, with additivity across features.

5.4.1 Univariate extensions: Polynomial regression

Polynomial regression is a method that extends linear regression by introducing polynomial terms into the model. It assumes that the relationship between the response y_i and the feature x_i can be represented as a polynomial of degree d :

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_d x_i^d + \epsilon_i$$

Here, ϵ_i is the error term, and the feature x_i is assumed to belong to some input space.

Remark.

This model can be fitted using OLS, Ridge, or Lasso methods.

In polynomial regression, while the individual coefficients $\beta_0, \beta_1, \ldots, \beta_d$ may not be easily interpretable, the main interest lies in the overall trend captured by the fitted polynomial function:

$$\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \cdots + \hat{\beta}_d x^d, \quad \forall x \in \mathcal{X}$$

Remark.

In practice, the degree d of the polynomial is typically chosen to be no greater than 4, and the best degree is often determined via cross-validation.

Definition 5.18: Logit

Mathematically, the logit is the inverse of the standard logistic function $\sigma(x) = 1/(1+e^{-x})$, so the logit is defined as

$$\operatorname{logit} p = \sigma^{-1}(p) = \log \frac{p}{1-p} \quad \text{for} \quad p \in (0,1)$$

Example.

Polynomial regression method is not only useful for regression but can also be applied to classification problems. For example, in logistic regression, the logit of the probability $\mathbb{P}(Y_i = 1 \mid X_i = x_i)$ can be modeled as a polynomial of x_i :

$$\operatorname{logit}\left(\mathbb{P}(Y_i = 1 \mid X_i = x_i)\right) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_d x_i^d$$

The coefficients in this case are fitted by maximizing the likelihood function.

Remark.

However, one important drawback of polynomial regression is that it exhibits poor behavior at the tails, meaning it performs poorly when extrapolating beyond the range of the observed data.

5.4.2 Univariate extensions: Step function

In contrast to polynomial regression, which imposes a global structure on the nonlinearity of X by fitting a continuous polynomial curve, the step function approach breaks the range of X into discrete intervals, or bins.

This method avoids the assumption of a single global structure across the entire feature space by instead allowing for different behaviors in different intervals of X. By discretizing the feature space, step functions provide more flexibility in capturing local patterns within the data, especially when the relationship between Y and X is not smooth or is highly variable across different ranges of X.

The step function approach works by partitioning the range of X into K pre-specified cut points $c_1, c_2, \ldots, c_K$, where:

$$c_1 \leq c_2 \leq \cdots \leq c_{K-1} \leq c_K$$

These cut points divide X into $K+1$ intervals, and for each interval, we define an indicator or dummy variable $C_j(X)$, where $C_j(X)$ takes the value of 1 if X falls within a particular interval, and 0 otherwise. Specifically, we have:

$$\begin{aligned} C_0(X) &= 1\{X < c_1\} \\ C_1(X) &= 1\{c_1 \leq X < c_2\} \\ &\vdots \\ C_K(X) &= 1\{c_K \leq X\} \end{aligned}$$

Remark.

Here, $C_0(X), C_1(X), \ldots, C_K(X)$ are dummy variables that indicate whether the feature X falls into a specific interval. Importantly, these $(K+1)$ indicator variables sum to 1 for any value of X, ensuring that X is assigned to exactly one interval.

The step function regression model assumes that the response Y depends on the step functions $C_j(x_i)$ as follows:

$$y_i = \beta_0 + \beta_1 C_1(x_i) + \beta_2 C_2(x_i) + \ldots + \beta_K C_K(x_i) + \epsilon_i$$

Here, ϵ_i represents the error term, and y_i is the observed response for the i-th data point. The coefficients $\beta_0, \beta_1, \ldots, \beta_K$ are estimated using techniques such as ordinary least squares (OLS) or shrinkage methods like Lasso or Ridge regression.

Remark.

> In this model, the intercept term β_0 captures the baseline effect, and thus, we do not include $C_0(x_i)$ in the model since its effect is absorbed into β_0.

Remark.

> The coefficient β_j in the step function model represents the average change in the response variable Y for values of X in the interval $[c_j, c_{j+1})$, relative to the baseline interval $X < c_1$. In other words, β_j measures how much the response changes in a specific bin or range of the feature X, compared to the baseline bin.

Pros and cons for step function method

1. **Pros:**
   - *Simplicity:* The step function approach is straightforward to understand and implement. It is widely used in fields like biostatistics and epidemiology due to its ease of interpretation.
   - *Interpretability:* The regression coefficients in the step function model have a natural interpretation. Each coefficient corresponds to the change in the response variable for a specific range of X, making it easy to explain the effects of different intervals.
2. **Cons:**
   - *Loss of Trend Information:* A major drawback of step functions is that they model the relationship between Y and X as piecewise constant. This means they can miss important trends or patterns in the data that are continuous or gradually changing across the feature space. For instance, if the true relationship between Y and X is smooth, a step function model may fail to capture this smoothness and instead produce abrupt changes at the cut points.
   - *Choice of Cut Points:* Specifying the appropriate cut points $c_1, c_2, \ldots, c_K$ can be difficult. Poorly chosen cut points may result in a model that either underfits or overfits the data. This is a key limitation, as the model's flexibility depends on the number and location of these cut points.

5.4.3 Univariate extensions: Piecewise polynomial

Given the strengths and limitations of both polynomial regression and step functions, one might consider combining these approaches. By blending the local flexibility of step functions with the smoothness and global structure of polynomials, it may be possible to capture both the local patterns and the overall trend in the relationship between Y and X.

A piecewise polynomial approach allows the model to use different polynomials in different regions of the input space X, as opposed to fitting a single global polynomial over the entire range of X. For instance, the model might fit one cubic polynomial to the data when X is less than some cutoff point c, and a different cubic polynomial when X is greater than or equal to c :

$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \epsilon_i & \text{if } x_i < c \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \epsilon_i & \text{if } x_i \geq c \end{cases}$$

Here, c is called a knot, and more knots can be introduced to make the piecewise polynomial model more flexible. If K different knots are placed throughout the range of X, then the model fits ($K+1$) cubic polynomials, each in its own region. This flexibility allows the model to capture local variations in the relationship between Y and X.

5.4.4 Univariate extensions: Regression splines

Definition 5.19: Basis function

The concept of basis functions is central to the construction of splines and many other types of regression models. In a basis representation, the model is written as a linear combination of basis functions $b_k(x_i)$:

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_K b_K(x_i) + \epsilon_i$$

The basis functions $b_k(\cdot)$ can take various forms, depending on the nature of the model.

While piecewise polynomials offer flexibility, they can produce abrupt transitions at the knots. Regression splines improve upon piecewise polynomials by adding constraints at the knots to ensure continuity and smoothness of the fitted function. The constraints can ensure:

- **Continuity:** The function values are equal at each knot.
- **Smoothness:** The first and second derivatives (or higher) are continuous at each knot.

When these constraints are applied, the resulting piecewise polynomial is called a spline. A degree- d spline consists of piecewise polynomials of degree d, with continuity in the derivatives up to degree $(d-1)$ at each knot. For example, a cubic spline (degree 3) will have continuous first and second derivatives at each knot.

A linear spline is the simplest form of a spline, using piecewise linear functions that are continuous at each knot. If there are K knots at positions $\xi_1 < \xi_2 < \cdots < \xi_K$, the model takes the following form:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 (x_i - \xi_1)_+ + \cdots + \beta_{K+1} (x_i - \xi_K)_+ + \epsilon_i$$

Here, $(x_i - \xi_k)_+$is a piecewise function defined as:

$$(x_i - \xi_k)_+ = \begin{cases} x_i - \xi_k & \text{if } x_i > \xi_k \\ 0 & \text{otherwise} \end{cases}$$

In this model, β_1 represents the average change in Y for a one-unit increase in X when $X < \xi_1$. This piecewise linear formulation allows the model to capture different linear trends in different regions of X.

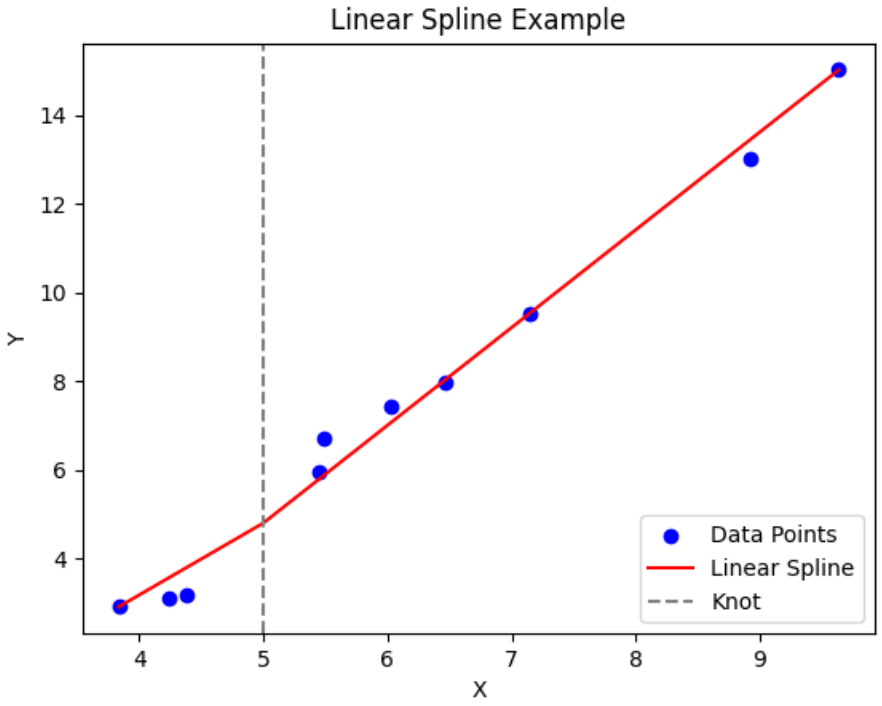

Figure 6: Example of a linear spline

A cubic spline uses piecewise cubic polynomials, with continuous first and second derivatives at each knot. This ensures that the model remains smooth at the points where the polynomials meet. With K knots at $\xi_1 < \xi_2 < \cdots < \xi_K$, the cubic spline model can be expressed as:

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i$$

The basis functions for cubic splines include:

$$b_1(x_i) = x_i, \quad b_2(x_i) = x_i^2, \quad b_3(x_i) = x_i^3$$
$$b_{k+3}(x_i) = (x_i - \xi_k)_+^3, \quad k = 1, \ldots, K$$

These basis functions ensure that the model fits cubic polynomials in each region and maintains smooth transitions at the knots.

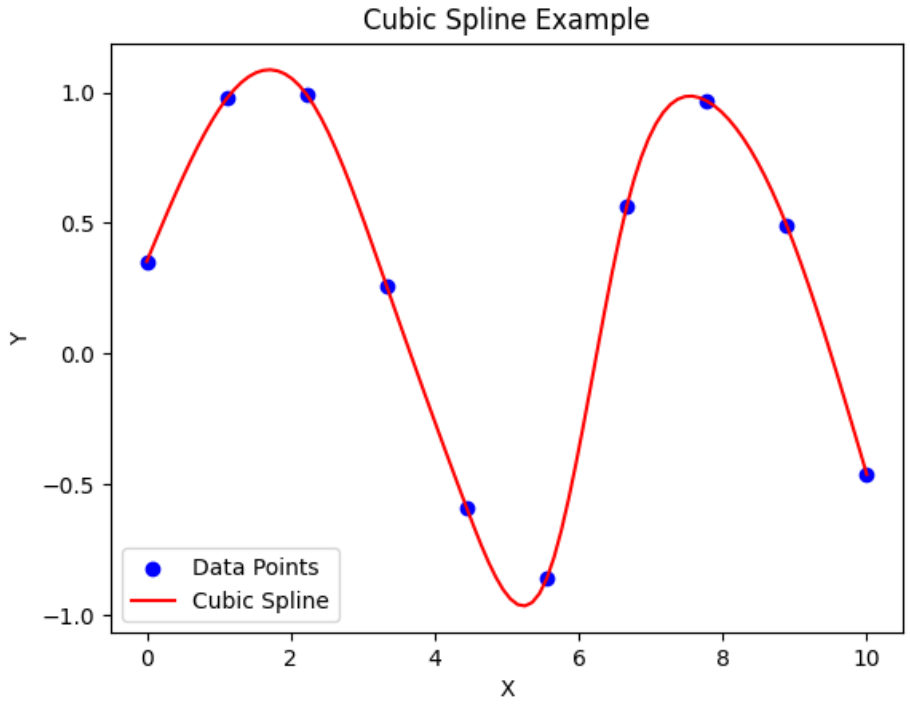

Figure 7: Example of a cubic spline

A natural spline is a special type of regression spline that imposes additional constraints at the boundaries of the data. Specifically, it forces the function to be linear at the boundaries, which can improve the model's behavior in cases where extrapolation beyond the range of the observed data is necessary. Natural splines maintain the flexibility of splines while controlling their behavior in extreme regions, making them useful in many practical applications.

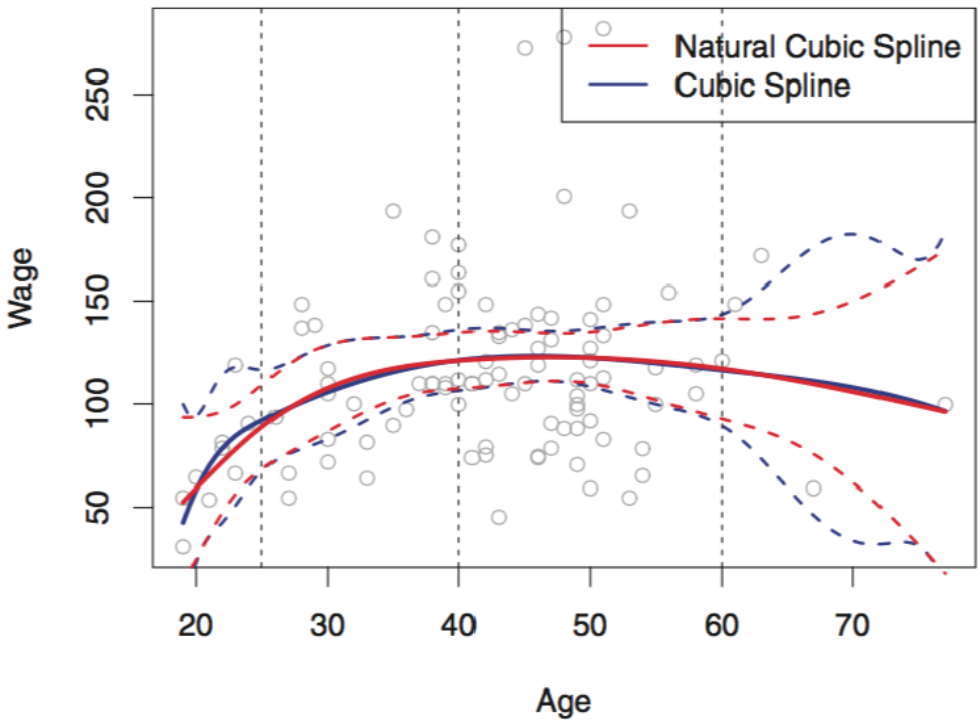

Figure 8: Example of a natural spline

- **Cubic Spline:** Extends freely with increasing curvature beyond the outermost knots, which can lead to unrealistic extrapolation, especially noticeable in the behavior past the age of 60 and before 20.
- **Natural Cubic Spline:** Remains linear beyond the outermost knots. This constraint provides a more stable extrapolation, as the function does not wildly fluctuate outside the range of the data. It flattens out as it approaches the boundaries of the data set, leading to potentially more reliable predictions in those regions.

Remark.

When using splines, choosing the number and location of the knots is important but not overly sensitive. Typically, knots are placed at quantiles of the data or at equally spaced points across the range of X. The exact placement of knots is often not crucial to the performance of the model, but increasing the number of knots leads to greater flexibility. The optimal number of knots can be selected using cross-validation, which helps balance model flexibility with generalizability.

Remark.

It is also important to note that both polynomial regressions and step functions are special cases of splines. In polynomial regression, there are no internal knots, while in step functions, the polynomials are degree-zero, meaning they are constant over each interval.

5.4.5 Multivariate extensions: Local Approaches for $p < 4$ using k-NN

For a relatively small number of predictors (fewer than four), local modeling techniques like nearest neighbor approaches and local regression are particularly effective. These methods focus on capturing the behavior of the response variable around the vicinity of the points of interest, allowing for a flexible fit that adapts to the local structure of the data.

Example.

The k-nearest neighbors algorithm is a simple yet powerful non-parametric method used both for classification and regression. Here's how it works in a regression context:

1. **Selecting k:** Choose k, the number of neighbors to consider. This can be any number from 1 to n,

where n is the total number of available data points.

2. **Finding Neighbors:** For a prediction at a new point $X = x_0$, identify the k nearest neighbors of x_0 among the existing data points $\{x_1, \ldots, x_n\}$. This subset is denoted as $\mathcal{N}_k(x_0)$.

3. **Predicting the Response:** Calculate the predicted response $\hat{f}(x_0)$ by averaging the responses of the k nearest neighbors:

$$\hat{f}(x_0) = \frac{1}{k} \sum_{i \in \{1,\ldots,n\}: x_i \in \mathcal{N}_k(x_0)} y_i$$

Example.

The choice of k has a significant impact on the performance of the model:

- $k = 1$: Using only the nearest neighbor leads to a very flexible model with high variance and low bias. The model closely follows the training data, but it may overfit, particularly in noisy datasets.
- $k = 3$: A slightly larger k reduces variance by smoothing over more points and slightly increases bias.
- $k = 10$: A much larger k averages over more neighbors, thus providing a smoother and less flexible model. This increases bias but significantly reduces variance.

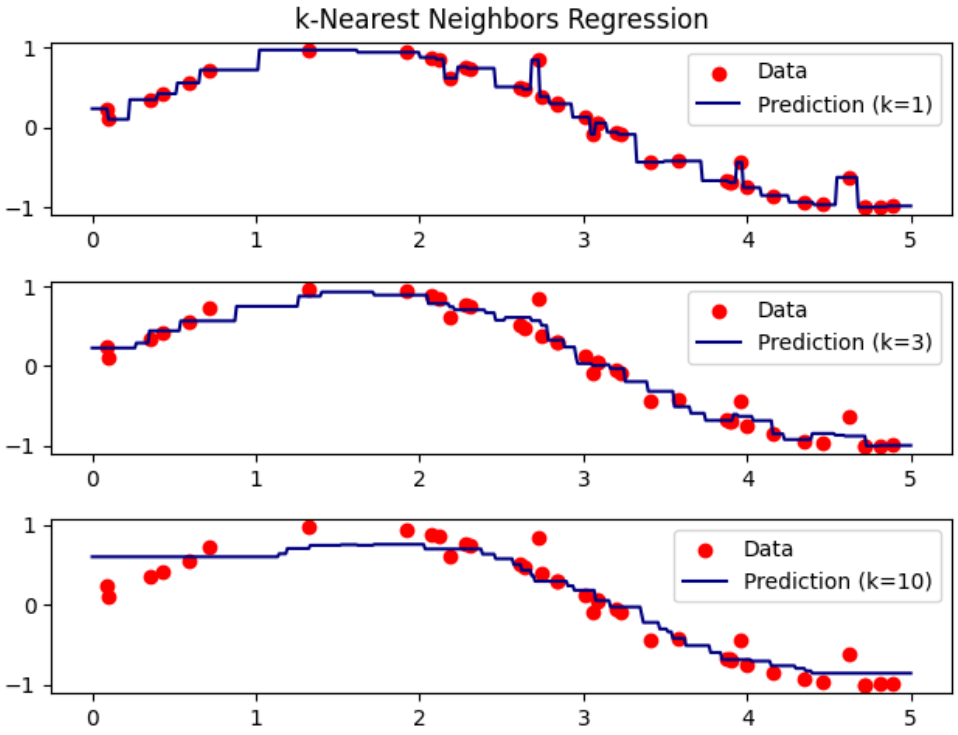

Figure 9: Different k nearest neighbor regression

Remark.

The choice of k controls the bias-variance tradeoff:

- A smaller k offers a more flexible model that closely captures local variations but at the risk of overfitting and higher variance.
- A larger k smooths out noise and provides more stable predictions but at the cost of potentially missing finer details in the data (higher bias).

Remark.

> The optimal value of k is typically selected through cross-validation. This involves dividing the data into several subsets, using different subsets for training and validation, and choosing the k that minimizes the validation error. This approach helps ensure that the model generalizes well to new, unseen data.

5.4.6 Generalization of k-NN: weighted k-NN

The k-nearest neighbors algorithm can be generalized by introducing weighted voting, where not all neighbors contribute equally to the final prediction. In standard k - NN , each of the k neighbors contributes equally to the prediction, but in weighted k - NN , closer neighbors can have more influence than those farther away.

Weighted k-NN formulation is doing the prediction at a point x_0 is given by:

$$\hat{f}(x_0) = \sum_{i \in \{1,\ldots,n\}: x_i \in \mathcal{N}_k(x_0)} K(x_i, x_0)\, y_i$$

Here, $K(x_i, x_0)$ are weights assigned to each neighbor, satisfying:

$$0 \leq K(x_i, x_0) \leq 1, \qquad \sum_{i \in \{1,\ldots,n\}: x_i \in \mathcal{N}_k(x_0)} K(x_i, x_0) = 1$$

A popular choice for the weights is inverse distance weighting, where the weight assigned to each neighbor is inversely proportional to its distance from the query point x_0. This means closer neighbors have a higher influence on the prediction. The weights are calculated as:

$$ID_i = \frac{1}{\|x_i - x_0\|_2}, \qquad \text{for all } x_i \in \mathcal{N}_k(x_0)$$

$$K(x_i, x_0) = \frac{ID_i}{\sum_{i: x_i \in \mathcal{N}_k(x_0)} ID_i}, \qquad \text{for all } x_i \in \mathcal{N}_k(x_0)$$

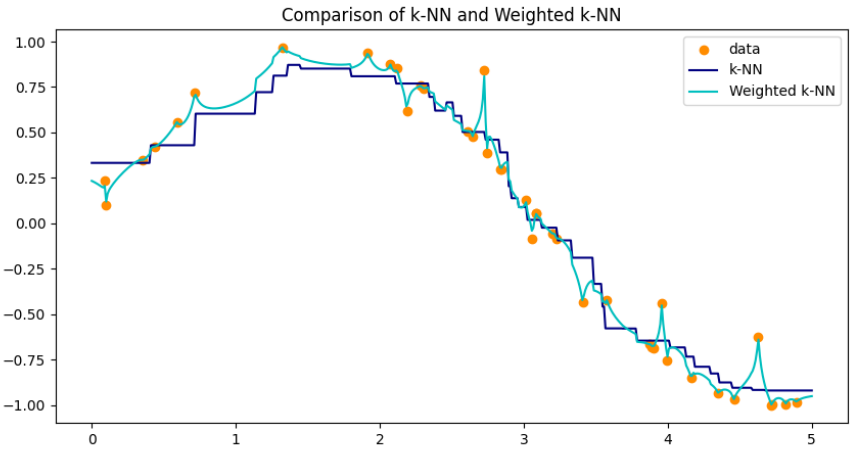

Figure 10: Regular k-NN and weighted k-NN

5.4.7 Generalization of k-NN: local regression

Local linear regression extends the idea of k - NN by not just averaging the outcomes of the neighbors but by fitting a linear model to these points. This approach allows the model to adapt to changes in the slope of the underlying function, providing a more flexible fit.

To predict at a target point x_0, the local linear regression method fits a linear model using only the nearby training observations, specifically those within the k-nearest neighbors:

$$\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$$

where $\hat{\beta}_0$ and $\hat{\beta}_1$ are estimated by minimizing the weighted least squares:

$$\left(\hat{\beta}_0, \hat{\beta}_1\right) = \underset{\beta_0, \beta_1}{\operatorname{argmin}} \sum_{i: x_i \in \mathcal{N}_k(x_0)} K\left(x_i, x_0\right)\left(y_i - \beta_0 - \beta_1 x_i\right)^2$$

Algorithm of local regression

1. **Selection of Neighbors:** Gather a fraction $s = k/n$ of training points whose x_i are closest to x_0.
2. **Weight Assignment:** Assign a weight $K_{i0} = K\left(x_i, x_0\right)$ to each point in this neighborhood.
3. **Model Fitting:** Fit a weighted least squares regression of the y_i on the x_i using these weights.
4. **Prediction:** The fitted value at x_0 is given by $\hat{f}\left(x_0\right) = \hat{\beta}_0 + \hat{\beta}_1 x_0$.

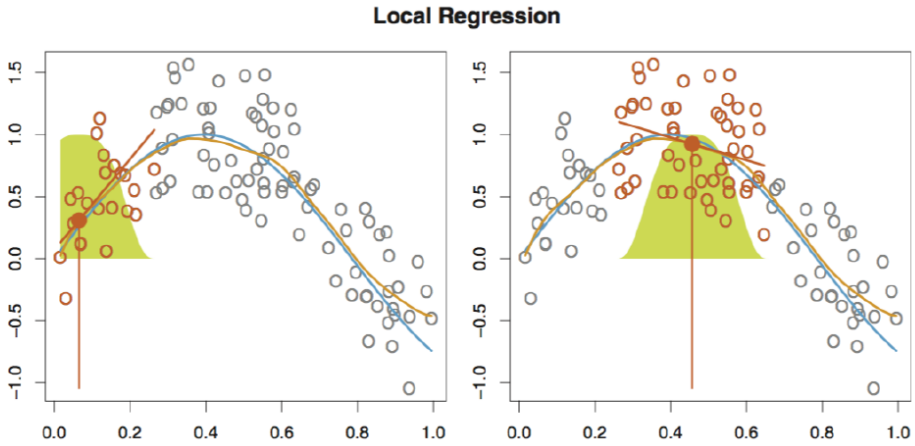

Figure 11: Illustration of local regression

The blue curve is true $f(x)$, and the light orange curve is the local regression $\hat{f}(x)$. The orange points are local to the target point x_0, represented by the orange vertical line. The yellow bell-shape indicates weights assigned to each point. The fit $\hat{f}\left(x_0\right)$ at x_0 is obtained by fitting a weighted linear regression (orange line segment), and using the fitted value at x_0 (orange solid dot) as the estimate $\hat{f}\left(x_0\right)$.

k-NN vs. local regression

1. **Flexibility:** Local linear regression provides a more flexible approach than k - NN by modeling changes in the slope of the relationship, not just the average outcome.
2. **Bias-Variance Tradeoff:** While both methods manage the bias-variance tradeoff, local linear regression typically offers better bias reduction at the cost of potentially higher variance due to the model complexity.
3. **Dimensionality:** Both methods suffer from the curse of dimensionality, where performance degrades as the number of predictors p increases beyond a small number (typically $p \geq 4$).

5.4.8 General additive model

Generalized Additive Models (GAMs) represent a significant extension of linear models by incorporating non-linear transformations of predictors while preserving the additive nature of the linear model framework.

A typical GAM can be described with the following equation:

$$y_i = \beta_0 + f_1\left(x_{i1}\right) + f_2\left(x_{i2}\right) + \cdots + f_p\left(x_{ip}\right) + \epsilon_i$$

- β_0 : Represents the intercept.

- $f_j(x_{ij})$: Each f_j is a smooth function that describes the effect of the j-th predictor x_{ij} on the response variable y_i. These functions f_j can be chosen based on the data type and the relationship observed between the predictors and the response variable. Common choices for f_j include polynomials, step functions, splines, and outputs from local regressions.
- ϵ_i : The error term, which is assumed to be normally distributed in many applications but can be modeled differently depending on the distribution of the response.

Remark.

GAMs can also be applied in classification scenarios through the use of a link function, such as the logistic function for binary classification:

$$\text{logit}\left(\mathbb{P}\left(Y_i = 1 \mid X_i = x_i\right)\right) = \beta_0 + f_1\left(x_{i1}\right) + f_2\left(x_{i2}\right) + \cdots + f_p\left(x_{ip}\right)$$

This makes GAMs flexible for handling various types of predictive modeling tasks, not just regression but also classification.

Pros and cons of GAM

1. **Pros:**
   - *Flexibility:* Allows each predictor to be modeled by a suitable function, capturing non-linear relationships without specifying a global form for the entire model.
   - *Interpretability:* Despite the non-linear components, the additive structure enables partial dependence plots and easy interpretation of each variable's effect on the outcome, holding other variables constant.
   - *Avoids Curse of Dimensionality:* By assuming additivity rather than interactions among predictors, GAMs sidestep the exponential growth of parameter space that plagues models like full non-linear regression.
2. **Cons:**
   - *No Interaction Terms:* By default, GAMs do not model interactions between predictors. While this simplifies interpretation and computation, it can miss important relationships in the data where the effect of one variable depends on the level of another.

6 Classification and logistic regression

6.1 Introduction to classification problem

In classification problems, the response variable Y is qualitative, meaning it takes values from an unordered set C, representing different classes or categories. The nature of the classification task depends on the number of unique values or classes in C :

- **Binary classification** occurs when $|C| = 2$, meaning there are only two possible classes.
- **Multi-class classification** occurs when $|C| > 2$, meaning there are more than two possible classes.

Given training data $\mathcal{D}^{\text{train}} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where each $y_i \in C$ and $x_i \in \mathbb{R}^p$ (the feature space), the goal in classification is threefold:

1. **Building a Classifier:** The classifier, represented as $\hat{f} : \mathbb{R}^p \to C$, is a function that assigns any future observation $x \in \mathbb{R}^p$ to a predicted class $\hat{f}(x) \in C$.
2. **Assessing Accuracy:** The accuracy of the classifier $\hat{f}$ must be evaluated, often using metrics like classification error or accuracy.
3. **Interpretation of Features:** We also want to understand how the different features x_i contribute to the classifier, which is key for interpretability and feature importance analysis.

To evaluate a classifier, we typically introduce a random pair (X, Y) that is independent of the training data $\mathcal{D}^{\text{train}}$. The class labels are encoded as $C = \{0, 1, 2, \ldots, K-1\}$, where K is the number of classes. For a classifier $\hat{f}$, its performance is measured by the expected error rate:

$$\mathbb{E}[1\{Y \neq \hat{f}(X)\}]$$

This expression represents the expected probability that the true label Y differs from the predicted label $\hat{f}(X)$. The challenge is to find the classifier that minimizes this error.

In the regression context, the best predictor is the regression function. For any given input $x \in \mathbb{R}^p$, the regression function is defined as:

$$f^*(x) = \mathbb{E}[Y \mid X = x]$$

This function minimizes the mean squared error (MSE), which is given by:

$$\mathbb{E}\left[(Y - \hat{f}(X))^2 \mid X = x\right]$$

The best predictor in regression is thus $f^*(x)$, and the corresponding minimum MSE is the irreducible error, $\sigma^2 = \text{Var}(\epsilon)$.

In the classification context, the optimal classifier is given by the Bayes classifier or Bayes rule. This classifier, denoted $f^* : \mathbb{R}^p \to C$, is the function that minimizes the expected classification error rate:

$$f^*(x) = \underset{f(x) \in C}{\text{argmin}} \mathbb{E}[1\{Y \neq \hat{f}(X)\} \mid X = x], \quad \forall x \in \mathbb{R}^p$$

In other words, for each observation x, the Bayes classifier assigns the most probable class label. The Bayes error rate is the lowest possible classification error that can be achieved, and it is given by:

$$\mathbb{E}\left[1\left\{Y \neq f^*(X)\right\}\right]$$

This error rate represents the minimum error any classifier can achieve because it is based on the true underlying distribution of the data.

Remark.

> The intuition behind this is the Bayes classifier $f^*(x)$ assigns a class label to x based on the conditional probability that Y belongs to that class, given $X = x$. Mathematically:
>
> $$f^*(x) = \arg\max_{k \in C} \mathbb{P}(Y = k \mid X = x)$$

Thus, the Bayes classifier selects the class with the highest posterior probability for a given observation x.

The Bayes error rate at a particular $x \in \mathbb{R}^p$ is calculated as:

$$\mathbb{E}\left[1\left\{Y \neq f^*(X)\right\} \mid X = x\right] = 1 - \max_{1 \leq j \leq K} \mathbb{P}(Y = j \mid X = x)$$

This means the Bayes error is 1 minus the probability of the most likely class for a given x. It is important to note that:

- The Bayes error rate lies between 0 and 1 .
- It typically does not equal 0 because real-world data is often noisy or overlapping between classes.

In the special case of binary classification where $C = \{0, 1\}$, the Bayes classifier is defined as:

$$f^*(x) = \begin{cases} 1, & \text{if } \mathbb{P}(Y = 1 \mid X = x) \geq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

Thus, in binary classification, the goal is to estimate the conditional probability $p(x) = \mathbb{P}(Y = 1 \mid X = x)$, which is a mapping from $\mathbb{R}^p$ to $\{0, 1\}$

Remark.

While it may seem natural to use a regression approach to estimate $\mathbb{P}(Y = 1 \mid X)$, there are limitations. Recall that in regression, the expected value is modeled as:

$$Y = f(X) + \epsilon = \mathbb{E}[Y \mid X] + \epsilon$$

In this case, we could use ordinary least squares (OLS) regression to estimate $\mathbb{E}[Y \mid X]$. However, OLS would predict probabilities using a linear model:

$$\hat{\beta}_0 + \hat{\beta}_1 X_1 + \cdots + \hat{\beta}_p X_p$$

This linear approach has limitations because it could predict probabilities outside the range $[0, 1]$. Therefore, while OLS can be used to estimate $p(X)$, a more tailored approach—such as logistic regression-is generally preferred for classification problems, as it constrains predictions to valid probability values between 0 and 1 .

6.2 Logistic regression in binary context

6.2.1 Basic definitions

In binary classification, the goal is to estimate the probability of the class being 1 , given the input $\mathbf{x}$. This probability is denoted as:

$$p(\mathbf{x}) := \mathbb{P}(Y = 1 \mid X = \mathbf{x})$$

- This means we are trying to find out how likely it is that the output $Y = 1$ for a given input $\mathbf{x}$.
- We can also express this as the expected value of Y given $\mathbf{X} = \mathbf{x}$, i.e., $p(\mathbf{x}) = \mathbb{E}[\mathbf{Y} \mid \mathbf{X} = \mathbf{x}]$, which reflects that the probability is the average outcome over many observations of similar inputs $\mathbf{x}$.

Definition 6.1: Logit function

The logistic function is written as:

$$f(t) = \frac{e^t}{1 + e^t}$$

- This function always outputs a value between 0 and 1 , which is why it's suitable for modeling probabilities.
- As $t \to -\infty, f(t) \to 0$, and as $t \to +\infty, f(t) \to 1$, so the function is bounded between 0 and 1.

Definition 6.2: Logistic regression

Logistic regression assumes that the probability $p(\mathbf{x})$ follows a specific functional form. Specifically, we model it using the logistic function:

$$p(\mathbf{x};\boldsymbol{\beta}) = \frac{e^{\beta_0+\beta_1 x_1+\cdots+\beta_p x_p}}{1+e^{\beta_0+\beta_1 x_1+\cdots+\beta_p x_p}}, \quad \forall \mathbf{x} \in \mathcal{X}$$

- Here, $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)$ are the parameters that we need to estimate from the data.
- $x_1, x_2, \ldots, x_p$ are the features of our input $\mathbf{x}$.

Definition 6.3: Log

The odds represent the ratio of the probability of the event happening $(p(\mathbf{x}))$ to the probability of it not happening $(1-p(\mathbf{x}))$:

$$\frac{p(\mathbf{x})}{1-p(\mathbf{x})} = e^{\beta_0+\beta_1 x_1+\cdots+\beta_p x_p}$$

Here, the odds range from 0 to infinity. If the odds are greater than 1 , the event is more likely to happen than not.

Definition 6.4: Log odd

Taking the natural logarithm of the odds gives us the log-odds or the logit:

$$\log\left(\frac{p(\mathbf{x})}{1-p(\mathbf{x})}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

The log-odds can take any value from $-\infty$ to $+\infty$, and the advantage of this transformation is that it turns the equation into a linear combination of the features, which is easier to interpret.

Remark.

In this linear equation, β_0 is the intercept and $\beta_1, \beta_2, \ldots, \beta_p$ are the coefficients corresponding to the features $X_1, X_2, \ldots, X_p$.
Each β_j represents the change in the log-odds for a one-unit increase in the feature X_j, holding all other features constant.
For example, if $\beta_1 = 0.5$, it means that for every one-unit increase in X_1, the log-odds of the event happening increase by 0.5 .

Goals of logistic regression

There are two main goals of logistic regression:

- One key goal of logistic regression is prediction. Given a new input $\mathbf{x}_0 \in \mathcal{X}$, we want to predict the corresponding label y_0. This is done by calculating the probability $p(\mathbf{x}_0)$ using the logistic function and deciding whether y_0 should be classified as 1 or 0 based on the probability.
- Another goal is estimation. We need to estimate the vector of parameters $\boldsymbol{\beta}$ from the training data. This is usually done through maximum likelihood estimation (MLE), where we find the values of $\boldsymbol{\beta}$ that make the observed data most likely.

Consider the estimated parameters, $\hat{\boldsymbol{\beta}} = \left(\hat{\beta}_0, \ldots, \hat{\beta}_p\right)$, which are the values of $\boldsymbol{\beta}$ that we estimate from the training data.

1. The logit function or log-odds is the linear part of the logistic regression model. Once we have estimates

of $\boldsymbol{\beta}$, the predicted logit at a given point $\mathbf{x} = (x_1, \ldots, x_p)$ is:

$$\hat{\text{logit}}(\mathbf{x}) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p$$

2. The next step is converting the logit to the actual probability $\hat{p}(\mathbf{x})$ of the event occurring (i.e., $Y = 1$), using the logistic function:

$$\hat{p}(\mathbf{x}) = \frac{e^{\hat{\beta}_1 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p}}$$

This is the predicted probability that the outcome $Y = 1$ given the input $\mathbf{x}$.

3. Finally, we need to classify the input $\mathbf{x}$ based on the predicted probability $\hat{p}(\mathbf{x})$. The classification rule is:

$$\hat{y} = \begin{cases} 1, & \text{if } \hat{p}(\mathbf{x}) \geq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

This means if the predicted probability $\hat{p}(\mathbf{x})$ is greater than or equal to 0.5 , we classify the input as belonging to class 1 . Otherwise, we classify it as class 0 .

This 0.5 threshold is a default for binary classification, but it can be adjusted depending on the problem.

6.2.2 Using MLE

Now we move into how we estimate the parameters $\boldsymbol{\beta}$ in logistic regression. One of the most common methods is using the Maximum Likelihood Estimation (MLE). We are given a training dataset $\mathcal{D}^{\text{train}} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, where each $y_i \in \{0, 1\}$ is the observed outcome (either class 0 or class 1) and each $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ represents the features for the i-th observation.

The idea behind MLE is to find the parameters $\boldsymbol{\beta}$ that maximize the likelihood of observing the given data. In other words, we want to find the set of parameters that makes the predicted probabilities $\hat{p}(\mathbf{x}_i)$ as close as possible to the actual outcomes y_i.

The likelihood function represents the probability of the observed data as a function of the parameters $\boldsymbol{\beta}$. In logistic regression, we aim to find the values of $\boldsymbol{\beta}$ that maximize this likelihood function.

General steps to MLE

1. **Write down the likelihood function:** This is a mathematical expression that represents the probability of the observed outcomes given the parameters.
2. **Solve the optimization problem:** Once we have the likelihood function, we maximize it with respect to the parameters $\boldsymbol{\beta}$. This can be done using numerical optimization techniques, as there is no closed-form solution for logistic regression.

For simplicity, we assume that $\beta_0 = 0$. This assumption simplifies the equations, though in real scenarios, β_0 (the intercept) is often not set to zero. Under this simplification, the probability of $Y = 1$ given $\mathbf{x}$, which is $p(\mathbf{x}; \boldsymbol{\beta})$, is given by:

$$p(\mathbf{x}; \boldsymbol{\beta}) = \frac{e^{\mathbf{x}^\top \boldsymbol{\beta}}}{1 + e^{\mathbf{x}^\top \boldsymbol{\beta}}}$$

The probability of $Y = 0$, which is $1 - p(\mathbf{x}; \boldsymbol{\beta})$, becomes:

$$1 - p(\mathbf{x}; \boldsymbol{\beta}) = \frac{1}{1 + e^{\mathbf{x}^\top \boldsymbol{\beta}}}$$

The data consists of observations $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, where each $y_i \in \{0, 1\}$ follows a Bernoulli distribution:

$$y_i \sim \text{Bernoulli}\left(p\left(\mathbf{x}_i; \boldsymbol{\beta}\right)\right)$$

This means that the response y_i is a binary variable, where the probability that $y_i = 1$ is $p\left(\mathbf{x}_i; \boldsymbol{\beta}\right)$, and the probability that $y_i = 0$ is $1 - p\left(\mathbf{x}_i; \boldsymbol{\beta}\right)$.

The likelihood of observing a single data point $(\mathbf{x}_i, y_i)$ depends on whether $y_i = 1$ or $y_i = 0$. The likelihood for a single data point $L(\boldsymbol{\beta}; \mathbf{x}_i, y_i)$ is proportional to:

$$L(\boldsymbol{\beta}; \mathbf{x}_i, y_i) \propto [p(\mathbf{x}_i; \boldsymbol{\beta})]^{y_i} [1 - p(\mathbf{x}_i; \boldsymbol{\beta})]^{1-y_i}$$

This formula reflects that:

- If $y_i = 1$, the likelihood is $p(\mathbf{x}_i; \boldsymbol{\beta})$.
- If $y_i = 0$, the likelihood is $1 - p(\mathbf{x}_i; \boldsymbol{\beta})$.

The joint likelihood is the product of the likelihoods of all individual data points:

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} [p(\mathbf{x}_i; \boldsymbol{\beta})]^{y_i} [1 - p(\mathbf{x}_i; \boldsymbol{\beta})]^{1-y_i}$$

The log-likelihood for logistic regression is:

$$\ell(\boldsymbol{\beta}) = \log \left(\prod_{i=1}^{n} [p(\mathbf{x}_i; \boldsymbol{\beta})]^{y_i} [1 - p(\mathbf{x}_i; \boldsymbol{\beta})]^{1-y_i} \right)$$

Using properties of logarithms, this simplifies to:

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} [y_i \log(p(\mathbf{x}_i; \boldsymbol{\beta})) + (1 - y_i) \log(1 - p(\mathbf{x}_i; \boldsymbol{\beta}))]$$

We can express the log-likelihood in a more compact form by expanding $(1 - y_i) \log(1 - p(\mathbf{x}_i; \boldsymbol{\beta}))$ and combine $-y_i \log(1 - p(\mathbf{x}_i; \boldsymbol{\beta}))$ with $y_i \log(p(\mathbf{x}_i; \boldsymbol{\beta}))$:

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[y_i \log \left(\frac{p(\mathbf{x}_i; \boldsymbol{\beta})}{1 - p(\mathbf{x}_i; \boldsymbol{\beta})} \right) + \log(1 - p(\mathbf{x}_i; \boldsymbol{\beta})) \right]$$

Using the fact that:

$$\log \left(\frac{p(\mathbf{x}_i; \boldsymbol{\beta})}{1 - p(\mathbf{x}_i; \boldsymbol{\beta})} \right) = \log \left(\frac{\frac{e^{\mathbf{x}_i^\top \beta}}{1+e^{\mathbf{x}_i^\top \beta}}}{1 - \frac{e^{\mathbf{x}_i^\top \beta}}{1+e^{\mathbf{x}_i^\top \beta}}} \right) = \mathbf{x}_i^\top \boldsymbol{\beta}$$

The log-likelihood becomes:

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[y_i \mathbf{x}_i^\top \boldsymbol{\beta} - \log \left(1 + e^{\mathbf{x}_i^\top \boldsymbol{\beta}} \right) \right]$$

This is the final form of the log-likelihood function for logistic regression. It is the function we need to maximize to find the optimal parameter vector $\boldsymbol{\beta}$.

Maximizing $\ell(\boldsymbol{\beta})$ is equivalent to minimizing the negative log-likelihood $-\ell(\boldsymbol{\beta})$. In many optimization problems, we prefer minimization. Hence, our goal becomes:

$$\min_{\beta} -\ell(\boldsymbol{\beta})$$

For some models (e.g., ordinary linear regression), we can find the exact solution to the maximum likelihood estimate by taking derivatives (the gradient) of the log-likelihood function, setting them to zero, and solving the resulting equations. Logistic regression doesn't allow such a closed-form solution. The reason is that the loglikelihood function is nonlinear and involves the term $e^{\mathbf{x}_i^\top \beta}$, which makes it impossible to solve for $\boldsymbol{\beta}$ directly after differentiation.

Since there's no explicit formula for $\boldsymbol{\beta}$, we use iterative optimization methods to find the values that maximize the log-likelihood function. The most common methods include:

- **Gradient descent:** This is a popular iterative technique where we take steps proportional to the negative of the gradient of the objective function (in this case, the negative log-likelihood).
- **Newton-Raphson or Fisher scoring:** These are more advanced iterative methods that use second-order information (the Hessian matrix) to update the parameter estimates.

Advantages of using MLE

- As the number of data points n increases (i.e., as $n \to \infty$), the MLE $\hat{\boldsymbol{\beta}}$ approaches the true parameter values $\boldsymbol{\beta}$. This property is known as consistency:

  $$\hat{\boldsymbol{\beta}} - \boldsymbol{\beta} \to 0 \quad \text{in probability as} \quad n \to \infty$$

  This means that, with enough data, the MLE will give us estimates that are very close to the true parameters of the model.

- Another important property is that the MLE, when computed from a large enough sample, follows a normal distribution:

  $$\sqrt{n}(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}) \to N(0, \Sigma) \quad \text{in distribution as} \quad n \to \infty$$

  This means that the MLE behaves like a normal (Gaussian) random variable when the sample size is large. The covariance matrix Σ describes the variability of $\hat{\boldsymbol{\beta}}$, and this normal approximation can be useful for constructing confidence intervals and conducting hypothesis tests.

- MLE is asymptotically efficient, meaning that it achieves the smallest possible variance among all asymptotically unbiased estimators. In other words, no other estimator can have a smaller variance in the limit as the sample size grows.

Disadvantages of MLE

- As mentioned earlier, there is no closed-form solution for β in logistic regression, which means we need to rely on iterative optimization algorithms. These algorithms can be computationally expensive, especially for large datasets or high-dimensional problems (many features).

  The convergence of iterative methods can be slow, and sometimes, the optimization can get stuck in local minima, especially if the log-likelihood surface is complex.

- The MLE relies on the assumption that the model is correctly specified. If the model is misspecified (for example, if we assume a logistic model but the true relationship is something else), the MLE can produce biased or incorrect estimates.

  In practice, we need to ensure that the logistic model is a good fit for the data; otherwise, MLE might not work well.

Remark.

For the section above, we assume the the intercept equals to 0 for simplicity. The probability including the intercept is:

$$p(\mathbf{x}; \boldsymbol{\beta}) = \frac{e^{\beta_0 + \mathbf{x}^\top \boldsymbol{\beta}}}{1 + e^{\boldsymbol{\beta}_0 + \mathbf{x}^\top \boldsymbol{\beta}}}$$

6.2.3 Inference with logistic regression

In logistic regression, after fitting the model and obtaining the maximum likelihood estimates (MLEs) for the parameters $\hat{\boldsymbol{\beta}}$, we are often interested in inference, specifically in testing whether individual coefficients β_j are statistically significant.

Definition 6.5: Z statistics in logistic regression

The Z -statistic is used to assess the significance of each coefficient β_j, analogous to the t statistic in linear regression. It is calculated as:

$$Z_j = \frac{\hat{\beta}_j}{\mathrm{SE}\left(\hat{\beta}_j\right)}$$

- $\hat{\beta}_j$ is the estimated coefficient from the logistic regression model.
- $\mathrm{SE}\left(\hat{\beta}_j\right) = \sqrt{\hat{\Sigma}_{jj}/n}$, where $\hat{\Sigma}_{jj}$ is the variance of $\hat{\beta}_j$ and n is the sample size. $\mathrm{SE}\left(\hat{\beta}_j\right)$ is the standard error of $\hat{\beta}_j$, which is derived from the asymptotic variance. The standard error helps measure the uncertainty of the estimate $\hat{\beta}_j$.

The Z-statistic is used to test the null hypothesis:

$$H_0 : \beta_j = 0 \quad \text{vs.} \quad H_1 : \beta_j \neq 0$$

A large absolute value of the Z-statistic or a small p-value provides evidence against the null hypothesis, suggesting that the predictor is statistically significant (i.e., it has a meaningful impact on the outcome).

The p-value gives the probability of observing a Z-statistic as extreme as the one calculated if the null hypothesis were true. A small p-value (commonly below 0.05) indicates that the null hypothesis can be rejected, implying that the coefficient β_j is likely non-zero and the corresponding predictor X_j has an effect on the outcome.

Example.

Consider the following default data. The goal is to predict the probability of default for a customer, using their student status as the only predictor. The model is specified as:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X$$

Here:

- $X = 1$ if the customer is a student, and $X = 0$ otherwise.
- $Y = 1$ if the customer defaults, and $Y = 0$ otherwise.

The predicted probability of default for a given student status $X = x$ is:

$$p(x) = \mathbb{P}(Y = 1 \mid X = x) = \frac{e^{\beta_0+\beta_1 x}}{1 + e^{\beta_0+\beta_1 x}}$$

The fitted maximum likelihood estimates (MLEs) for the coefficients are provided in the table:

Coefficient	Estimate	Std. Error	Z-statistic	P-value
Intercept	-3.5	0.071	-49.55	< 0.0001
student[Yes]	0.405	0.115	3.52	0.0004

Table 1: Result of the MLE

For students $(x = 1)$:

$$\hat{p}(x = 1) = \frac{e^{-3.5+0.405\times 1}}{1 + e^{-3.5+0.405\times 1}} \approx 0.043$$

This means that the estimated probability of default for a student is approximately 4.3%. For non-students $(x = 0)$:

$$\hat{p}(x = 0) = \frac{e^{-3.5+0.405\times 0}}{1 + e^{-3.5+0.405\times 0}} \approx 0.029$$

The estimated probability of default for a non-student is approximately 2.9%.

Example.

Given the similar context of the previous example, we now extend our data. Now, we extend the model to include multiple predictors: balance (X_1), income (X_2), and student status (X_3). The model is:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3$$

Here, X_1 represents the customer's balance, X_2 represents their income, and $X_3 = 1$ if the customer is a student.
The MLEs for the coefficients are provided in the table:

Coefficient	Estimate	Std. Error	Z-statistic	P-value
Intercept	-10.87	0.492	-22.08	< 0.0001
balance	0.006	0.0002	24.74	< 0.0001
income	0.003	0.0082	0.37	0.712
student[Yes]	-0.647	0.2362	-2.74	0.0062

Table 2: Extended model and its results

- **Balance** (X_1)**:** The coefficient for balance is positive and highly significant (Z-statistic $= 24.74$, p value < 0.0001). This suggests that as the customer's balance increases, the probability of default increases.
- **Income** (X_2)**:** The coefficient for income is not statistically significant (Z-statistic $= 0.37$, p-value $= 0.712$), suggesting that income does not have a meaningful impact on the probability of default in this model.
- **Student Status** (X_3)**:** The coefficient for student status is negative and significant (Z-statistic = -2.74 , p -value $= 0.0062$). This suggests that being a student actually decreases the probability of default when controlling for balance and income.

In the first model, where student status was the only predictor, the coefficient for student status was positive ($\beta_1 = 0.405$). This indicated that students had a higher probability of default compared to non-students.
In the second model, where balance and income were also included, the coefficient for student status became negative ($\beta_3 = -0.647$). This indicates that, after controlling for balance and income, being a student actually decreases the probability of default.

6.2.4 Evaluating binary classifiers

In classification, we have multiple metrics to assess how well a model performs. While accuracy is the most commonly used metric, especially in binary classification, there are many other metrics that give us a better understanding of the model's performance.

Definition 6.6: Overall classification accuracy

Accuracy is the proportion of correct predictions (both positive and negative) out of the total predictions made. It's calculated as:

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{Total predictions}}$$

However, accuracy can be misleading in imbalanced datasets (e.g., if one class dominates). And we have more options for binary classifications.

We are trying to classify whether an individual will default on their credit card payment based on two predictors: credit card balance and student status. The confusion matrix for the fitted logistic regression model looks like this:

		True default status		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

Table 3: Confusion matrix

- **True positives (TP):** 81 - the model correctly predicts that 81 individuals will default.
- **True negatives (TN):** 9,644 - the model correctly predicts that 9,644 individuals will not default.

- **False positives (FP):** 23 - the model incorrectly predicts that 23 individuals will default when they won't.
- **False negatives (FN):** 252 - the model incorrectly predicts that 252 individuals won't default when they actually do.

From this confusion matrix, we can calculate the training error rate as:

$$\text{Training error rate } = \frac{\text{FP} + \text{FN}}{\text{Total observations}} = \frac{23 + 252}{10,000} = 2.75\%$$

This training error rate tells us that 2.75% of the classifications are incorrect.

Definition 6.7: False positive rate FPR

FPR is the proportion of negative examples (those who will not default) that are incorrectly classified as positive (defaulting):

$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}} = \frac{23}{9,667} = 0.2\%$$

A small FPR indicates that the model is good at avoiding false alarms (i.e., wrongly classifying individuals who won't default as defaulters).

Definition 6.8: False negative rate FNR

FNR is the proportion of positive examples (those who will default) that are incorrectly classified as negative (non-defaulting):

$$\text{FNR} = \frac{\text{FN}}{\text{TP} + \text{FN}} = \frac{252}{333} = 75.7\%$$

A high FNR means that the model is missing a significant number of defaulters. In the context of credit card default, a high FNR is problematic because the model fails to identify many high-risk individuals who are likely to default. For the credit card company, this is unacceptable.

To reduce the false negative rate (FNR), we need to adjust how the classifier makes predictions. The current classification rule is:

$$\hat{y}_i = \begin{cases} 1 & \text{(default) ,} \quad \text{if } \hat{\mathbb{P}}(\text{ default } = \text{ yes } \mid X = x_i) \geq 0.5 \\ 0 & \text{(non-default),} \quad \text{otherwise.} \end{cases}$$

To lower the false negative rate, we need to reduce the number of negative predictions (i.e., we want to classify more individuals as defaulters, even if their predicted probability is less than 0.5). We can do this by lowering the classification threshold t, where:

$$\hat{\mathbb{P}}(Y = \text{ yes } \mid X = \mathbf{x}) \geq t$$

Instead of using $t = 0.5$, we can lower t to a value between 0 and 0.5 . This makes the classifier more likely to predict a default, thus catching more true defaulters, but potentially increasing the number of false positives.

- The threshold $t = 0.5$ is commonly used because it represents the midpoint of probability. If we predict that an event is more likely than not, we classify it as positive. However, in many cases, especially when false negatives are costly (as with credit defaults), a lower threshold might be preferred.
- If $t = 0$, we classify everyone as defaulting. This would result in 0 false negatives but a huge number of false positives. It's an extreme solution that eliminates missed defaulters but leads to many people incorrectly classified as defaulters.
- If $t = 1$, we classify no one as defaulting, because the predicted probability would never reach 1 for anyone in practice. This would result in 0 false positives but a massive number of false negatives (you'd miss all defaulters).

In binary classification, there is often a trade-off between the False Positive Rate (FPR) and the False Negative Rate (FNR), depending on the threshold t used for classification:

- **A lower threshold (e.g., below 0.5)** leads to more predictions of the positive class (higher true positives but also more false positives).
- **A higher threshold (e.g., above 0.5)** leads to fewer predictions of the positive class (reducing false positives but potentially increasing false negatives).

Definition 6.9: ROC curve (Receiver operating characteristic curve)

The ROC curve is a graphical representation that plots the True Positive Rate (TPR) (or sensitivity, which is $1 - FNR$) against the False Positive Rate (FPR) at various threshold settings.
The AUC (Area Under the Curve) of the ROC summarizes the overall performance of the classifier across all thresholds:

- AUC$= 1$ indicates a perfect classifier.
- AUC$= 0.5$ suggests a random classifier (no better than flipping a coin).
- Higher AUC values are better because they indicate a higher TPR for a given FPR across a range of thresholds.

In summary, we have the general format of the confusion matrix as below:

		Predicted class		
		- or Null	+ or Non-null	Total
True class	- or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N *	P*	

Table 4: General format of confusion matrix

Metric	Formula	Synonyms
False Positive Rate (FPR)	$\frac{\mathrm{FP}}{N}$	Type I error, 1 - Specificity
True Positive Rate (TPR)	$\frac{\mathrm{TP}}{P}$	Sensitivity, recall, power, 1 - Type II error
Positive Predictive Value (PPV)	$\frac{\mathrm{TP}}{P*}$	Precision, 1 - False Discovery Proportion
Negative Predictive Value (NPV)	$\frac{\mathrm{TN}}{N*}$	

Table 5: Summary of matrices

6.3 Gradient descent

6.3.1 General problem of optimization

The general optimization problem is:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}\in\Theta}{\operatorname{argmin}} \mathcal{J}\left(\mathbf{w}; \mathcal{D}^{\text{train}}\right)$$

We are searching for a value of the vector $\mathbf{w}$ (denoted by $\hat{\mathbf{w}}$) that minimizes a function $\mathcal{J}\left(\mathbf{w}; \mathcal{D}^{\text{train}}\right)$.

- w is a vector of parameters (weights) we are trying to optimize. These could be parameters of a model, like the coefficients in linear regression.
- $\mathcal{D}^{\text{train}}$ represents the training data, which influences $\mathcal{J}(\mathbf{w})$.
- $\mathcal{J}(\mathbf{w})$ is the objective function or loss function, a measure of how well our model fits the data. The goal is to minimize this function.
- Θ is the parameter space. In most cases, it is a subset of $\mathbb{R}^p$, which means we are optimizing in a p-dimensional real space (with p being the number of parameters).

Remark.

The objective function is assumed to be differentiable, meaning we can take derivatives of it with respect to the parameters $w_1, w_2, \ldots, w_p$.

The optimal solution (if it exists) must be a critical point, meaning it occurs where the derivatives of the function $\mathcal{J}(\mathbf{w})$ with respect to all parameters $w_1, \ldots, w_p$ are zero. Mathematically, for multi-dimensional problems, we have:

$$\begin{bmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_p} \end{bmatrix} = 0$$

In simpler terms, we calculate the partial derivatives (rate of change of the function with respect to each parameter) and set them equal to zero to find the minima.

For a function $\mathcal{J}(w_1, w_2)$, the partial derivative with respect to w_1 is computed as follows:

$$\frac{\partial}{\partial w_1} \mathcal{J}(w_1, w_2) = \lim_{h \to 0} \frac{\mathcal{J}(w_1 + h, w_2) - \mathcal{J}(w_1, w_2)}{h}$$

Once we have the partial derivatives for all the parameters, setting them equal to zero forms a system of linear equations. If this system is simple enough, we can solve it analytically (i.e., find an exact solution). However, for complex systems, **iterative optimization techniques** (such as gradient descent) are typically used to find the solution.

Example.

Consider the example of OLS estimator. We can solve the equation analytically. In OLS, the goal is to minimize the following objective function:

$$\hat{\mathbf{w}} = \underset{\mathbf{w} \in \mathbb{R}^p}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$$

The partial derivative of the OLS objective function with respect to the weight vector $\mathbf{w}$ is:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = -2\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w})$$

This derivative represents the gradient of the loss function. Setting it to zero and solving for $\mathbf{w}$ gives us the solution for the optimal parameters:

$$\mathbf{X}^\top \mathbf{X} \hat{\mathbf{w}} = \mathbf{X}^\top \mathbf{y}$$

The closed-form solution for the weights in OLS is:

$$\hat{\mathbf{w}} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

This is the standard formula used to compute the optimal weights in linear regression.

Example.

The objective function for Ridge Regression is:

$$\hat{\mathbf{w}}_\lambda^R = \underset{\mathbf{w} \in \mathbb{R}^p}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_2^2$$

Here, $\lambda \|\mathbf{w}\|_2^2$ is the regularization term. This penalizes large values of the weights (discouraging overly complex models). The gradient of the Ridge Regression objective function is:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = -2\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda \mathbf{w}$$

Setting the gradient to zero and solving for $\mathbf{w}$ gives:

$$\left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_p\right) \hat{\mathbf{w}}_\lambda^R = \mathbf{X}^\top \mathbf{y}$$

This leads to the solution:

$$\hat{\mathbf{w}}_\lambda^R = \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_p\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

6.3.2 Gradient descent: Alternative solution

The goal remains the same:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \mathcal{J}(\mathbf{w})$$

We want to find the value of $\mathbf{w}$ that minimizes the function $\mathcal{J}(\mathbf{w})$, our loss or objective function. In many cases, we can't solve this minimization problem directly (i.e., we can't solve $\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = 0$ analytically), especially for complex or high-dimensional problems. This is where gradient descent becomes useful.

Definition 6.10: Gradient descent

Gradient descent is an iterative algorithm for finding the minimum of a function. Instead of solving for the minimum in one step (as we did with OLS), gradient descent makes gradual adjustments to the parameters $\mathbf{w}$, and it "descends" towards the minimum step by step.
Gradient descent follows some general key steps:

1. **Initialization:** We start with some initial value for $\mathbf{w}$, which can be chosen randomly or initialized to zero (or any reasonable value).
2. **Iterative Updates:** At each step, we adjust $\mathbf{w}$ in the direction of the steepest descent, which is the direction that decreases the objective function $\mathcal{J}(\mathbf{w})$ the fastest. The direction of the steepest descent is given by the negative gradient of the objective function at the current value of $\mathbf{w}$.

Definition 6.11: The graident of the function and direction

The gradient of the function $\mathcal{J}(\mathbf{w})$ at a point $\mathbf{w}$ is a vector of partial derivatives with respect to each parameter in w. Mathematically:

$$\left.\frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{(0)}} \in \mathbb{R}^p$$

This gradient vector tells us the direction of steepest ascent at the current point $\mathbf{w}^{(0)}$, i.e., the direction in which $\mathcal{J}(\mathbf{w})$ increases the most.
Since we want to minimize the function (not maximize it), we move in the opposite direction of the gradient. This is the key idea behind gradient descent.

Definition 6.12: Update rule for gradient descent

To adjust the parameters $\mathbf{w}$, we apply the following update rule:

$$\mathbf{w}^{(1)} = \mathbf{w}^{(0)} - \alpha \cdot \left.\frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{(0)}}$$

- $\mathbf{w}^{(1)}$ is the new value of the parameters after one step of gradient descent.
- $\mathbf{w}^{(0)}$ is the current value of the parameters.
- α is the learning rate, a small positive number that controls how large a step we take in the direction of the negative gradient.
- $\frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}}$ is the gradient of the loss function at the current value $\mathbf{w}^{(0)}$.

Definition 6.13: Learning rate α

The parameter $\alpha > 0$ is called the learning rate, and it plays a crucial role in how fast or slow the algorithm converges to the minimum.

- If α is too small, gradient descent will take very small steps and thus will converge very slowly.

- If α is too large, the algorithm might overshoot the minimum and fail to converge, or even diverge (i.e., the objective function $\mathcal{J}(\mathbf{w})$ could keep increasing rather than decreasing).

If α is chosen appropriately, the algorithm guarantees that:

$$\mathcal{J}\left(\mathbf{w}^{(1)}\right) < \mathcal{J}\left(\mathbf{w}^{(0)}\right)$$

This means that after one update, the objective function's value at the new parameters $\mathbf{w}^{(1)}$ is smaller than it was at $\mathbf{w}^{(0)}$, assuming the gradient at $\mathbf{w}^{(0)}$ is not zero.

We can breakdown the gradient descent coordinate wise. For each iteration $k = 0, 1, 2, \ldots$, the update happens coordinate-wise, meaning we update each parameter w_j (where $j = 1, 2, \ldots, p$) individually:

$$w_j^{(k+1)} \leftarrow w_j^{(k)} - \alpha \cdot \left.\frac{\partial \mathcal{J}}{\partial w_j}\right|_{\mathbf{w}=\mathbf{w}^{(k)}}$$

This means for each parameter w_j, we adjust its value based on the gradient of the objective function $\mathcal{J}$ with respect to that parameter at the current step k. The key elements are:

- $w_j^{(k)}$: the current value of the parameter w_j at iteration k.
- α : the learning rate, which determines how large of a step we take in the direction of the negative gradient. A larger α leads to bigger changes in w_j, but if it's too large, the algorithm might overshoot the minimum.
- $\frac{\partial \mathcal{J}}{\partial w_j}$: the partial derivative of $\mathcal{J}$ with respect to w_j, which tells us how sensitive the objective function is to changes in w_j.

Thus, at each iteration, the parameters are updated one by one in a stepwise manner, and we repeat this until convergence.

Example.

Here, we can apply gradient descent to OLS estimators:

$$\mathcal{J}(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$$

This is the least-squares loss function, which measures the sum of squared residuals between the true values $\mathbf{y}$ and the predicted values $\mathbf{X}\mathbf{w}$ (where $\mathbf{X}$ is the matrix of input features and $\mathbf{w}$ is the vector of weights).
The update rule for gradient descent in vector form (for all parameters $\mathbf{w}$) at iteration $k+1$ is:

$$\mathbf{w}^{(k+1)} \leftarrow \mathbf{w}^{(k)} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}}$$

This means that for each iteration, we adjust the entire vector $\mathbf{w}$ in the direction of the negative gradient of $\mathcal{J}(\mathbf{w})$. In OLS, the partial derivative of $\mathcal{J}(\mathbf{w})$ with respect to $\mathbf{w}$ is:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = -2\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w})$$

Substituting this into the update rule, we get:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + 2\alpha\mathbf{X}^\top\left(\mathbf{y} - \mathbf{X}\mathbf{w}^{(k)}\right) = \hat{\mathbf{w}}^{(k)} + 2\alpha\sum_{i=1}^{n}\left[y_i - \mathbf{x}_i^\top\hat{\mathbf{w}}^{(k)}\right]\mathbf{x}_i.$$

This means that at each iteration, the new value of $\mathbf{w}$ is the old value of $\mathbf{w}$, plus a step in the direction of the gradient (scaled by the learning rate α).
We typically start by initializing $\mathbf{w}^{(0)} = 0$, which means that we start with all weights being zero and then iteratively adjust them using gradient descent.

Stopping criteria for gradient descent

Since gradient descent is an iterative algorithm, we need some criteria to determine when to stop the iterations. There are a few common stopping criteria:

- **When the objective value stops changing:**

  We can stop when the difference between the objective function values at successive iterations is very small, meaning the algorithm has converged to a solution:

  $$\left|\mathcal{J}\left(\mathbf{w}^{(k+1)}\right) - \mathcal{J}\left(\mathbf{w}^{(k)}\right)\right| \leq 10^{-6}$$

  This ensures that the improvement in the objective function is so small that continuing further iterations would not result in a significantly better solution.

- **When the parameters stop changing:**

  Another stopping criterion is when the change in the parameters $\mathbf{w}$ between successive iterations becomes small. There are two ways to measure this:

  1. $\left\|\mathbf{w}^{(k+1)} - \mathbf{w}^{(k)}\right\|_2$ is small, meaning the Euclidean distance between the parameters from one iteration to the next is very small.
  2. $\left\|\mathbf{w}^{(k+1)} - \mathbf{w}^{(k)}\right\|_2 / \left\|\mathbf{w}^{(k)}\right\|_2$ is small, meaning the relative change in the parameters is small. This takes into account the scale of the parameters to ensure that small changes are meaningful relative to their current values.

- **Maximum number of iterations:**

  Finally, we can stop after a fixed number of iterations, M, even if the algorithm has not fully converged. For example, we might set $M = 1000$, meaning the algorithm will stop after 1000 iterations regardless of whether the other stopping criteria are met. This is useful to prevent the algorithm from running indefinitely if it doesn't converge in a reasonable time.

Example.

Now, we apply gradient descent to solve the Maximum Likelihood Estimation (MLE) problem under logistic regression. In logistic regression, the goal is to estimate the parameters (weights) $\mathbf{w}$ that maximize the loglikelihood function $\ell(\mathbf{w})$, or equivalently, minimize the negative log-likelihood (denoted here as $\mathcal{J}(\mathbf{w})$). The objective is to minimize the following function:

$$\min_{\mathbf{w}\in\mathbb{R}^p} \mathcal{J}(\mathbf{w})$$

where $\mathcal{J}(\mathbf{w})$ is the negative log-likelihood:

$$\mathcal{J}(\mathbf{w}) = -\ell(\mathbf{w}) = \sum_{i=1}^{n}\left[-y_i\mathbf{x}_i^\top\mathbf{w} + \log\left(1 + e^{\mathbf{x}_i^\top\mathbf{w}}\right)\right]$$

The partial derivative of this function with respect to w_j is:

$$\frac{\partial\mathcal{J}(\mathbf{w})}{\partial w_j} = \sum_{i=1}^{n}\left[\frac{\partial}{\partial w_j}\left(-y_i\mathbf{x}_i^\top\mathbf{w}\right) + \frac{\partial}{\partial w_j}\log\left(1 + e^{\mathbf{x}_i^\top\mathbf{w}}\right)\right]$$

The first term involves $\mathbf{x}_i^\top\mathbf{w} = \sum_{k=1}^{p} x_{ik}w_k$, and its derivative with respect to w_j is simply x_{ij}, the j-th feature for the i-th data point:

$$\frac{\partial}{\partial w_j}\left(-y_i\mathbf{x}_i^\top\mathbf{w}\right) = -y_i x_{ij}$$

The second term is the derivative of the $\log\left(1 + e^{\mathbf{x}_i^\top\mathbf{w}}\right)$ term. Using the chain rule:

$$\frac{\partial}{\partial w_j}\log\left(1 + e^{\mathbf{x}_i^\top\mathbf{w}}\right) = \frac{1}{1 + e^{\mathbf{x}_i^\top\mathbf{w}}} \cdot \frac{\partial}{\partial w_j}\left(1 + e^{\mathbf{x}_i^\top\mathbf{w}}\right)$$

The derivative of $e^{\mathbf{x}_i^\top \mathbf{w}}$ with respect to w_j is:

$$\frac{\partial}{\partial w_j} e^{\mathbf{x}_i^\top \mathbf{w}} = e^{\mathbf{x}_i^\top \mathbf{w}} x_{ij}$$

Thus, the full derivative of the second term is:

$$\frac{\partial}{\partial w_j} \log\left(1 + e^{\mathbf{x}_i^\top \mathbf{w}}\right) = \frac{e^{\mathbf{x}_i^\top \mathbf{w}}}{1 + e^{\mathbf{x}_i^\top \mathbf{w}}} x_{ij}$$

Putting the two terms together, we get the full expression for the partial derivative of $\mathcal{J}(\mathbf{w})$ with respect to w_j :

$$\frac{\partial \mathcal{J}(\mathbf{w})}{\partial w_j} = \sum_{i=1}^{n} \left[-y_i x_{ij} + \frac{e^{\mathbf{x}_i^\top \mathbf{w}}}{1 + e^{\mathbf{x}_i^\top \mathbf{w}}} x_{ij} \right]$$

Factor out x_{ij} :

$$\frac{\partial \mathcal{J}(\mathbf{w})}{\partial w_j} = \sum_{i=1}^{n} \left[-y_i + \frac{e^{\mathbf{x}_i^\top \mathbf{w}}}{1 + e^{\mathbf{x}_i^\top \mathbf{w}}} \right] x_{ij}$$

Since $\mathcal{J}(\mathbf{w}) = -\ell(\mathbf{w})$, we have:

$$-\frac{\partial \ell(\mathbf{w})}{\partial w_j} = \frac{\partial \mathcal{J}(\mathbf{w})}{\partial w_j}$$

Thus, the gradient is computed as follows for any $j \in \{1, 2, \ldots, p\}$:

$$-\frac{\partial \ell(\mathbf{w})}{\partial w_j} = \sum_{i=1}^{n} \left[-y_i + \frac{e^{\mathbf{x}_i^\top \mathbf{w}}}{1 + e^{\mathbf{x}_i^\top \mathbf{w}}} \right] x_{ij}$$

At the $k+1$-th iteration, the update rule for the weight vector $\mathbf{w}$ is:

$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} - \alpha \sum_{i=1}^{n} \left[-y_i + \frac{e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}}{1 + e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}} \right] \mathbf{x}_i = \hat{\mathbf{w}}^{(k)} + \alpha \sum_{i=1}^{n} \left[y_i - \frac{e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}}{1 + e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}} \right] \mathbf{x}_i$$

In this case, the weights are initialized to zero:

$$\mathbf{w}^{(0)} = 0$$

Remark.

In linear regression, even though we still have closed form solution, we still prefer gradient descent when the number of features p is large as it avoids the computationally expensive matrix inversion required by the direct solution in linear regression.

Remark.

Gradient descent can be applied to both linear and logistic regression, and in more complex machine learning models, gradient descent is often the go-to optimization method.

6.3.3 Conditions of gradient descent

Definition 6.14: Convex set

$\mathcal{S}$ is convex if for any $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{S}, (1-\lambda)\mathbf{x}_0 + \lambda \mathbf{x}_1 \in \mathcal{S}$ for all $0 \leq \lambda \leq 1$
This means that for any two points in the set, any point along the line connecting them must also be within the set.

Definition 6.15: Convex function

A function f is convex if for any two points $\mathbf{x}_0$ and $\mathbf{x}_1$ in the domain of f, the following condition holds:

$$f\left((1-\lambda)\mathbf{x}_0+\lambda\mathbf{x}_1\right) \leq (1-\lambda)f\left(\mathbf{x}_0\right)+\lambda f\left(\mathbf{x}_1\right), \forall\lambda\in[0,1]$$

This inequality means that the value of the function at any point on the line connecting $\mathbf{x}_0$ and $\mathbf{x}_1$ is less than or equal to the weighted average of the function's values at $\mathbf{x}_0$ and $\mathbf{x}_1$. Visually, this condition ensures that the function is bowl-shaped.

We have two key conditions for gradient descent to work:

- **Differentiability:** The objective function $\mathcal{J}(\mathbf{w})$ must be differentiable, which means we can compute its gradient at any point. Without this, we can't apply gradient descent in the way it has been described (using the gradient to update the weights).
- **Convexity:** If $\mathcal{J}(\mathbf{w})$ is a convex function, and the parameter space Θ is a convex set, then gradient descent is guaranteed to find the global minimum, provided a suitable step size (learning rate) is chosen.

Remark.

In many machine learning problems, the parameter space Θ is typically $\mathbb{R}^p$, which is convex, making gradient descent a viable optimization method for many models.

Remark.

A convex function has the property that any local minimum is also a global minimum. This is important because it ensures that gradient descent won't get stuck in a local minimum, and we can be confident that the solution it converges to is optimal.

Ways to determine if a function is convex

- You can check the mathematical definition of convexity, as mentioned above, by verifying the inequality condition for any two points in the domain of the function.
- For twice-differentiable functions, a simpler way to check convexity is by examining the second derivative:
  - If $f''(x) \geq 0$ for all x, then the function is convex.
  - For multi-dimensional functions, we use the Hessian matrix (the matrix of second-order partial derivatives). If the Hessian is positive semi-definite (i.e., all its eigenvalues are non-negative), then the function is convex.
- Certain composition rules help us determine if more complex functions are convex. Linear functions preserve convexity. If f is a convex function and g is a linear function, then both the compositions $f \circ g$ and $g \circ f$ are convex.

  The least squares loss $\left(y-\mathbf{x}^\top\mathbf{w}\right)^2$ is convex in $\mathbf{w}$, because the quadratic function is convex and the linear function $\mathbf{x}^\top\mathbf{w}$ preserves this convexity.

  The negative log-likelihood in logistic regression $-y\mathbf{x}^\top\mathbf{w}+\log\left(1+e^{\mathbf{x}^\top\mathbf{w}}\right)$ is convex in $\mathbf{w}$. This is because both terms individually are convex, and their sum is also convex.
- There are additional composition rules for checking convexity, especially for more complex functions. These rules help extend convexity from simpler functions to more complicated combinations of functions.

6.3.4 Effect of learning rate

The learning rate α is a key hyperparameter in the gradient descent algorithm. It determines the size of the steps taken towards the minimum of the cost function. The slides and the associated diagrams illustrate the consequences of different choices for α:

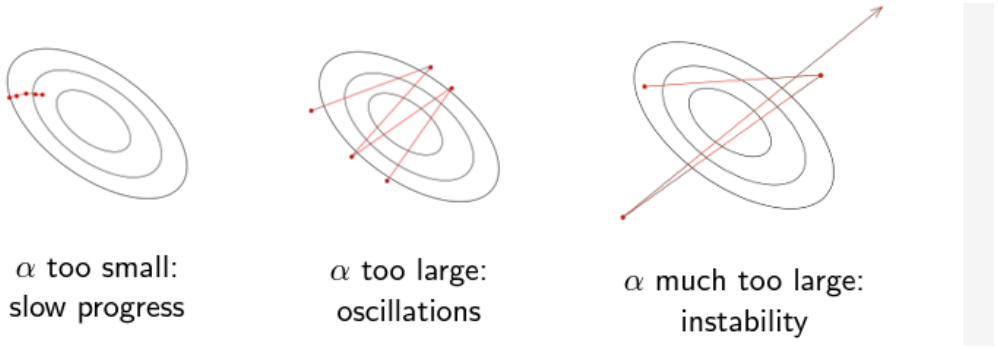

Figure 12: Various learning rates

- α **too small:** This results in very slow progress towards the minimum. The steps taken in each iteration of gradient descent are too small, leading to many iterations being necessary to reach the minimum, which can be computationally expensive and time-consuming.
- α **too large:** When the learning rate is too large, the updates may overshoot the minimum, causing the algorithm to oscillate around the minimum without settling. This can prevent convergence.
- α **much too large:** If the learning rate is excessively high, it can cause drastic updates that lead to diverging from the minimum entirely, resulting in an unstable optimization process where the cost may increase rather than decrease.

To find a good value of α, it's recommended to perform a grid search where multiple values of α are tested. Typical values to try might include $0.1, 0.03, 0.01$, and so on, decreasing by factors of about 3 or 10.

6.3.5 Traning curve

Training curves are graphical representations that plot the training cost (value of the objective function) as a function of iteration number. They are used to diagnose the behavior of the optimization algorithm over time.

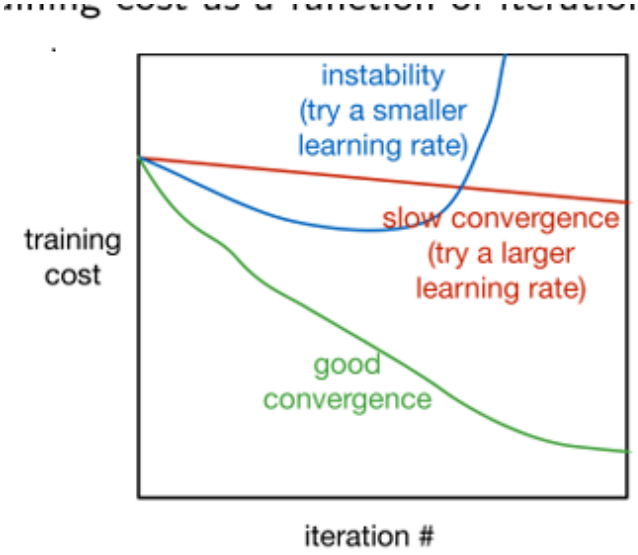

Figure 13: Training cost curve

- **Stable and Good Convergence:** Shows the cost decreasing smoothly and steadily as iterations proceed. This indicates that the learning rate and other hyperparameters are well-tuned.

- **Slow Convergence:** If the curve decreases too slowly, it might suggest that the learning rate is too small, and the algorithm is taking unnecessarily small steps towards the minimum.
- **Instability:** A curve that shows increasing cost or dramatic fluctuations might indicate that the learning rate is too high, causing instability in the learning process.

Several limitations of the training curve

- **Global Minimum:** The training curve alone cannot confirm whether the reached point is a global minimum, especially if the function is not convex.
- **Model Performance:** Lower training costs do not necessarily translate to better performance on unseen data. This could be due to overfitting, where the model learns the noise in the training data rather than the underlying pattern.

6.3.6 Batch gradient descent

Definition 6.16: Batch gradient descent

Batch Gradient Descent computes the gradient of the cost function using the entire dataset at each iteration. This is typically used in contexts where the dataset is not excessively large, and it's feasible to compute the gradients over the entire data set at once.

Recall we have the following formula:

- **OLS:**

$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} + 2\alpha \sum_{i=1}^{n} \left[y_i - \mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)} \right] \mathbf{x}_i.$$

- **Logistic regression:**

$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} + \alpha \sum_{i=1}^{n} \left[y_i - \frac{e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}}{1 + \mathrm{e}^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}} \right] \mathbf{x}_i.$$

What we did before are all batch gradient descent. The advantage of batch gradient descent is its straightforward implementation and stable error gradients. However, its disadvantage is that it can be very slow and computationally expensive when n (number of samples) is large, as each step involves computations over the entire dataset.

6.3.7 Stochastic gradient descent (SGD)

Definition 6.17: Stocahstic gradient descent

Stochastic Gradient Descent updates the parameters using only a single training example at each iteration. This is selected randomly, which significantly reduces the computational burden per iteration, making it suitable for very large datasets.

Both updates for OLS and logistic regression take the form:

- **For OLS:**

$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} + \alpha \left[y_i - \mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)} \right] \mathbf{x}_i$$

- **For Logistic Regression:**

$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} + \alpha \left[y_i - \frac{e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}}{1 + e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}} \right] \mathbf{x}_i$$

Advantages and disadvantages of SDG

Advantages:

- *Efficiency:* Each update is very fast, making the algorithm suitable for large datasets.
- *Convergence Speed:* It can make significant progress toward the minimum even before seeing the entire dataset, which can be beneficial in streaming or online learning scenarios.
- **Mathematical justification:** the gradients between SGD and GD have the same expectation for i.i.d. data.

Disadvantages of SGD:

- *High Variance:* Updates are noisy due to using only one data point at a time, which can cause the algorithm to bounce around the minimum rather than steadily converging.

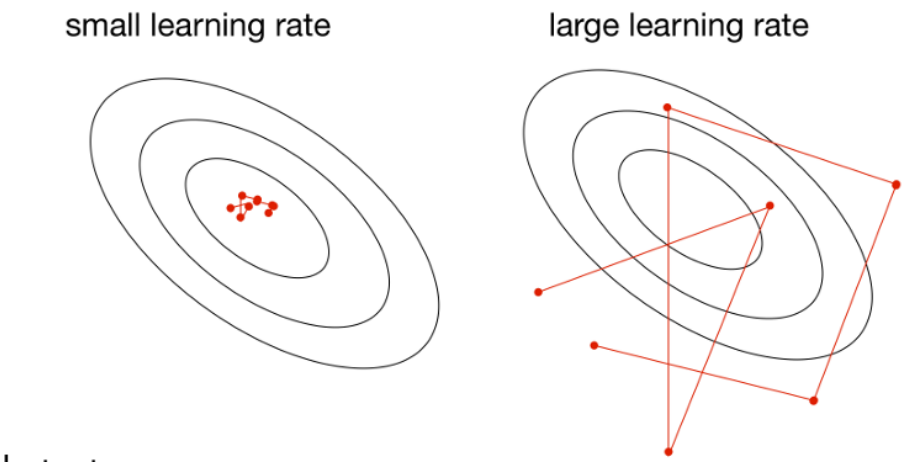

Figure 14: SGD learning rate

With a small learning rate, SGD shows slow but steady progress with smaller deviations from the path, whereas with a large learning rate, SGD takes larger, more erratic steps, which may sometimes lead to overshooting or diverging from the optimal path.

We have some strategies:

- Use a large learning rate early in training so you can get close to the optimum.
- Gradually decay the learning rate to reduce the fluctuations.

6.3.8 Mini-batch gradient descent

Definition 6.18: Mini-batch gradient descent

Mini-batch Gradient Descent is a compromise between batch and stochastic gradient descent. It uses a small subset of the training data $\mathcal{M} \subset \{1, \ldots, n\}$, called a mini-batch, to compute the gradient at each step.

6.3.9 Batch GD vs. SGD

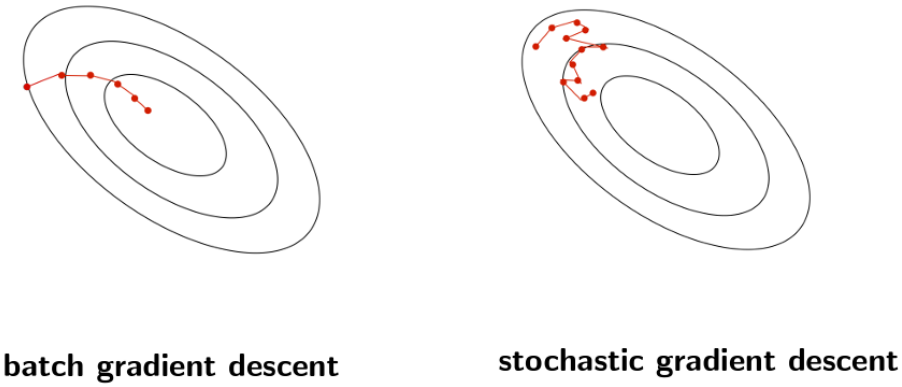

Figure 15: Trajectory of GD and SDG

- **For GD:** it moves directly towards the minimum, taking the steepest descent path as determined by the average of all gradients across the entire dataset. Because it uses all available data to compute the gradient, its path is smooth and it directly targets the global minimum (in convex scenarios) or local minimum (in non-convex scenarios).
- **For SGD:** Trajectory: it moves in a direction determined by the gradient of randomly selected individual samples. This introduces noise into the gradient calculation, resulting in a less direct path towards the minimum. Despite the noise in individual steps, over many iterations, the average direction of SGD's steps still points towards the minimum, making it effective on average.

6.4 Logistic regression in multi-class context

In real-world applications, classification tasks often involve more than two classes. When the target variable Y can take values from a set of K classes, i.e., $Y \in \{0, 1, \ldots, K-1\}$, we need to extend the logistic regression model to handle multi-class classification.

The goal now is to estimate the probability of Y belonging to each of the K classes. We define:

$$p_k(\mathbf{x}) := \mathbb{P}(Y = k \mid X = \mathbf{x}), \quad \forall 0 \leq k \leq K-1$$

This means we need to estimate the probability of each class k given the input $\mathbf{x}$.

Definition 6.19: Softmax regression (Multinomial logistic regression)

One common approach to multi-class classification is to use the softmax function. Here, we parametrize the conditional probabilities for each class k as follows:

$$p_0(\mathbf{x}) = \frac{1}{1 + \sum_{k=1}^{K-1} e^{\beta_0^{(k)} + \mathbf{x}^\top \boldsymbol{\beta}^{(k)}}}$$

$$p_k(\mathbf{x}) = \frac{e^{\beta_0^{(k)} + \mathbf{x}^\top \boldsymbol{\beta}^{(k)}}}{1 + \sum_{k=1}^{K-1} e^{\beta_0^{(k)} + \mathbf{x}^\top \boldsymbol{\beta}^{(k)}}}, \quad \forall k \geq 1$$

We can equivalently express the classification model in terms of log-odds, which is the logarithm of the ratio of probabilities between two classes (also called logits). Specifically, the log-odds of class k versus the baseline

class 0 are given by:

$$\log\left(\frac{p_1(\mathbf{x})}{p_0(\mathbf{x})}\right) = \beta_0^{(1)} + \beta_1^{(1)}x_1 + \cdots + \beta_p^{(1)}x_p$$
$$\log\left(\frac{p_2(\mathbf{x})}{p_0(\mathbf{x})}\right) = \beta_0^{(2)} + \beta_1^{(2)}x_1 + \cdots + \beta_p^{(2)}x_p$$
$$\vdots$$
$$\log\left(\frac{p_{K-1}(\mathbf{x})}{p_0(\mathbf{x})}\right) = \beta_0^{(K-1)} + \beta_1^{(K-1)}x_1 + \cdots + \beta_p^{(K-1)}x_p$$

The challenge in multi-class classification is estimating the coefficients $\boldsymbol{\beta}^{(k)}$ for each class. One naive approach to estimating these coefficients is to treat the problem as a series of separate binary logistic regressions, one for each class against a baseline.

The log-odds for class k versus the baseline class 0 is modeled as:

$$\log\left(\frac{p_k(\mathbf{x})}{p_0(\mathbf{x})}\right) = \beta_0^{(k)} + \beta_1^{(k)}x_1 + \cdots + \beta_p^{(k)}x_p$$

The naive approach suggests splitting the data and running separate binary logistic regression models for each class k. The steps are as follows:

1. **Split the Data:** For each class k (where $1 \leq k \leq K-1$), create a subset of the training data, $\mathcal{D}^{\text{train}}(k)$, which contains only the data points where the target variable Y is either 0 (the baseline) or k.

   Essentially, for each binary logistic regression model, the dataset will consist only of class 0 and class k.

2. **Binary Logistic Regression for Each Class:** For each binary logistic regression corresponding to class k, use the subset $\mathcal{D}^{\text{train}}(k)$ to estimate the coefficients $\boldsymbol{\beta}^{(k)}$.

   These estimates are used to compute the ratio $\frac{p_k(\mathbf{x})}{p_0(\mathbf{x})}$, which represents the likelihood of class k versus class 0 given the input features $\mathbf{x}$.

3. **Assign Class Label:** Once the models are trained, you can assign a class label to a new observation $\mathbf{x}$ by comparing the ratios:

   $$1, \frac{p_1(\mathbf{x})}{p_0(\mathbf{x})}, \frac{p_2(\mathbf{x})}{p_0(\mathbf{x})}, \ldots, \frac{p_{K-1}(\mathbf{x})}{p_0(\mathbf{x})}$$

   You would then assign the class that corresponds to the largest ratio.

This naive approach, while intuitive, has several important limitations:

1. **Limited Data for Estimation:** When estimating the coefficients $\boldsymbol{\beta}^{(k)}$, only the subset of the data points corresponding to classes 0 and k is used. This ignores potentially useful information from the other classes.

   In other words, the separate binary logistic regressions only use data from $\mathcal{D}^{\text{train}}(k)$, which is the data from the binary classification task involving 0 and k. However, this ignores the data from the other classes, which could help improve the estimation of $\beta^{(k)}$.

2. **Inter-class Dependencies Are Ignored:** In multi-class classification, the event that a given sample $y_i = k$ depends on the probabilities of it being classified into other classes (i.e., $y_i = k'$ for $k' \neq k$). By estimating $\boldsymbol{\beta}^{(k)}$ in isolation, we fail to account for these inter-class dependencies.

Instead of the naive approach, we should use the MLE method. The goal is to estimate the parameters $\boldsymbol{\beta}^{(k)}$ for each class k using MLE. The log-likelihood function quantifies how likely the observed data is given the parameters, and we want to maximize this likelihood with respect to $\boldsymbol{\beta}^{(k)}$.

The likelihood function $\mathcal{L}$ is the probability of observing the given labels $y_1, y_2, \ldots, y_n$ for the given input data $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, assuming that the probabilities are modeled by the softmax function. The likelihood is written as:

$$\mathcal{L}\left(\boldsymbol{\beta}^{(1)}, \ldots, \boldsymbol{\beta}^{(K-1)}\right) = \prod_{i=1}^{n}\prod_{k=0}^{K-1} p_k\left(\mathbf{x}_i\right)^{1\{y_i=k\}}$$

To simplify optimization, we maximize the log-likelihood function, which is the logarithm of the likelihood function. Taking the log of the likelihood:

$$\ell\left(\boldsymbol{\beta}^{(1)}, \ldots, \boldsymbol{\beta}^{(K-1)}\right) = \log\left(\prod_{i=1}^{n}\prod_{k=0}^{K-1} p_k\left(\mathbf{x}_i\right)^{1\{y_i=k\}}\right)$$

Using properties of logarithms, this becomes:

$$\ell\left(\boldsymbol{\beta}^{(1)}, \ldots, \beta^{(K-1)}\right) = \sum_{i=1}^{n}\sum_{k=0}^{K-1} 1\left\{y_i = k\right\} \log\left(p_k\left(\mathbf{x}_i\right)\right)$$

Next, substitute the expressions for $p_k(\mathbf{x})$ into the log-likelihood function. For the baseline class $k = 0$

$$p_0(\mathbf{x}) = \frac{1}{1 + \sum_{k=1}^{K-1} e^{\mathbf{x}_i^\top \boldsymbol{\beta}^{(k)}}}$$

and for $k \geq 1$:

$$p_k(\mathbf{x}) = \frac{e^{\mathbf{x}_i^\top \boldsymbol{\beta}^{(k)}}}{1 + \sum_{k=1}^{K-1} e^{\mathbf{x}_i^\top \boldsymbol{\beta}^{(k)}}}$$

Substitute these into the log-likelihood expression:

$$\begin{aligned}\ell\left(\boldsymbol{\beta}^{(1)}, \ldots, \ \boldsymbol{\beta}^{(K-1)}\right) = \sum_{i=1}^{n}\Bigg[1\left\{y_i = 0\right\} \log\left(\frac{1}{1 + \sum_{k=1}^{K-1} e^{\mathbf{x}_i^\top \boldsymbol{\beta}^{(k)}}}\right) \\ + \sum_{k=1}^{K-1} 1\left\{y_i = k\right\} \log\left(\frac{e^{\mathbf{x}_i^\top \boldsymbol{\beta}^{(k)}}}{1 + \sum_{k=1}^{K-1} e^{\mathbf{x}_i^\top \boldsymbol{\beta}^{(k)}}}\right)\Bigg]\end{aligned}$$

Simplify the logarithms:

$$\begin{aligned}= \sum_{i=1}^{n}\Bigg[1\left\{y_i = 0\right\}\left(-\log\left(1 + \sum_{k=1}^{K-1} e^{\mathbf{x}_i^\top \boldsymbol{\beta}^{(k)}}\right)\right) \\ + \sum_{k=1}^{K-1} 1\left\{y_i = k\right\}\left(\mathbf{x}_i^\top \boldsymbol{\beta}^{(k)} - \log\left(1 + \sum_{k=1}^{K-1} e^{\mathbf{x}_i^\top \boldsymbol{\beta}^{(k)}}\right)\right)\Bigg]\end{aligned}$$

Finally, collect terms to obtain the final form of the log-likelihood function:

$$\ell\left(\boldsymbol{\beta}^{(1)}, \ldots, \boldsymbol{\beta}^{(K-1)}\right) = \sum_{i=1}^{n}\left[\sum_{k=1}^{K-1} 1\left\{y_i = k\right\} \mathbf{x}_i^\top \boldsymbol{\beta}^{(k)} - \log\left(1 + \sum_{k=1}^{K-1} e^{\mathbf{x}_i^\top \boldsymbol{\beta}^{(k)}}\right)\right]$$

To maximize the log-likelihood, we need to compute the gradient of the log-likelihood function with respect to each parameter $\boldsymbol{\beta}^{(k)}$. The partial derivative of the log-likelihood with respect to $\boldsymbol{\beta}^{(k)}$ is:

$$\frac{\partial \ell\left(\boldsymbol{\beta}^{(1)}, \ldots, \boldsymbol{\beta}^{(K-1)}\right)}{\partial \boldsymbol{\beta}^{(k)}} = \sum_{i=1}^{n}\left[1\left\{y_i = k\right\} \mathbf{x}_i - \frac{\mathbf{x}_i e^{\mathbf{x}_i^\top \boldsymbol{\beta}^{(k)}}}{1 + \sum_{k=1}^{K-1} e^{\mathbf{x}_i^\top \boldsymbol{\beta}^{(k)}}}\right]$$

This can be rewritten as:

$$\frac{\partial \ell\left(\boldsymbol{\beta}^{(k)}\right)}{\partial \boldsymbol{\beta}^{(k)}} = \sum_{i=1}^{n}\left[1\left\{y_i = k\right\} - p_k\left(\mathbf{x}_i\right)\right] \mathbf{x}_i$$

In binary logistic regression (where $K = 2$), the gradient of the log-likelihood with respect to $\boldsymbol{\beta}$ is:

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{n}\left[1\left\{y_i = 1\right\} - \frac{e^{\mathbf{x}_i^\top \boldsymbol{\beta}}}{1 + e^{\mathbf{x}_i^\top \boldsymbol{\beta}}}\right] \mathbf{x}_i$$

This equation is structurally similar to the multi-class case, but it simplifies because there are only two classes.

And we have $\mathcal{J} = -\ell$, so, we have:

$$\frac{\partial \mathcal{J}}{\partial \beta^{(k)}} = -\sum_{i=1}^{n}\left[1\left\{y_i = k\right\} - p_k\left(\mathbf{x}_i\right)\right] \mathbf{x}_i$$

Using the gradient, we can update the parameters $\boldsymbol{\beta}^{(k)}$ using gradient descent. The update rule for the coefficients $\boldsymbol{\beta}^{(k)}$ is:

$$\hat{\boldsymbol{\beta}}_{(t+1)}^{(k)} = \hat{\boldsymbol{\beta}}_{(t)}^{(k)} + \alpha \sum_{i=1}^{n} \left[1\{y_i = k\} - \frac{e^{\mathbf{x}_i^\top \hat{\boldsymbol{\beta}}_{(t)}^{(k)}}}{1 + \sum_{k=1}^{K-1} e^{\mathbf{x}_i^\top \hat{\boldsymbol{\beta}}_{(t)}^{(k)}}} \right] \mathbf{x}_i$$

Remark.

> The gradient update uses data points from all classes, not just from a pair of classes at a time (as in the naive approach). This pooling of data allows for better estimation of the parameters $\boldsymbol{\beta}^{(k)}$ because the entire dataset is considered, capturing the inter-class relationships that the naive method misses.

6.5 Limitations of logistic regression

Logistic regression, while powerful, can have limitations, especially in certain situations. When the classes are well-separated, logistic regression can yield unstable parameter estimates. This is becuase when the classes are well-separated, the model can assign very high confidence to the class predictions. To achieve this, the model tries to drive the log-odds towards very large positive or negative values, meaning the parameters $\boldsymbol{\beta}$ (and possibly β_0) can grow arbitrarily large. In practice, the parameters can take on extremely large values, which means small changes in the data can lead to disproportionately large changes in the parameter estimates. This results in unstable estimates that are highly sensitive to noise or slight variations in the data.

In such cases, discriminant analysis can perform better. It tends to provide more stable estimates when classes are well-separated. It often outperforms logistic regression in multi-class classification problems because it models the entire data distribution and not just the conditional probability of the class given the features.

And when the sample size n is small and there is prior knowledge about the distribution of $X \mid Y = k$. Discriminant analysis is better.

7 Discriminant analysis

7.1 Introduction to discriminant analysis

As we have discuess before, we know that for logistic regression, it directly models the conditional probability $P(Y = k \mid X = x)$ for all classes $k \in C$. However, for discriminant analysis, it Models the distribution $P(X \mid Y = k)$, which represents the distribution of the features X given the class k.

Once $P(X \mid Y = k)$ is modeled, Bayes' theorem allows us to compute $P(Y = k \mid X = x\ )$, which is the probability of class k given the observed data $X = x$.

Remark.

> DA often assumes that $P(X \mid Y = k)$ follows a multivariate normal distribution, which simplifies computation and decision rules.

Theorem 7.1: Bayes' theorem in general

By this theorem, we know in general:

$$P(Y = k \mid X = x) = \frac{P(X = x \mid Y = k)P(Y = k)}{P(X = x)}$$

Intuitively, in order to classify $X = x$, we compare $P(Y = k \mid X = x)$ across all classes k. And using Bayes' rule:

$$P(Y = k \mid X = x) \geq P\left(Y = k' \mid X = x\right) \iff P(X = x \mid Y = k)P(Y = k) \geq P\left(X = x \mid Y = k'\right) P\left(Y = k'\right)$$

This reduces classification to comparing weighted likelihoods of $X = x$ for each class.

7.1.1 Notations in DA

In DA, suppose we have K classes, we denote them as $C = \{0, 1, 2, \ldots, K-1\}$. And for each class $k \in C$:

- Prior probability is:
$$\pi_k = P(Y = k)$$
This is the proportion of observations expected to belong to class k.
- we take conditional density function as:
$$f_k(x) = P(X = x \mid Y = k)$$
This models the likelihood of the features X for class k.
- Most importantly, $f_k(x)$ is assumed to follow a parametric distribution (e.g., normal distribution) for easier computation.

7.1.2 Bayes' rule application in DA

Using Bayes's rule from 7.1, we know that the posterior probability is given by:

$$p_k(x) = P(Y = k \mid X = x) = \frac{\pi_k f_k(x)}{\sum_{\ell \in C} \pi_\ell f_\ell(x)}$$

This formula calculates the probability of an observation belonging to class k given its features $X = x$.

And with such Bayes classifier, we assign $X = x$ to the class k with the highest posterior probability:

$$\textbf{Classify: } \arg\max_{k \in C} p_k(x)$$

The denominator $\sum_{\ell \in C} \pi_\ell f_\ell(x)$ as a constant normalizes the probabilities, ensuring they sum to 1.

Since $\sum_{\ell \in C} \pi_\ell f_\ell(x)$ is constant for all classes, the classifier simplifies to:

$$\arg\max_{k \in C} p_k(x) = \arg\max_{k \in C} \pi_k f_k(x)$$

This means the decision depends on the product of the prior π_k and likelihood $f_k(x)$.

7.2 DA and linear DA with $p = 1$ i.e. one feature

Assuming we only have one feature and **a univariate assumption** that $X \mid Y = k \sim \mathcal{N}(\mu_k, \sigma_k^2)$, which means X is normally distributed within each class k with mean μ_k and variance σ_k^2. The likelihood of $X = x$ given $Y = k$ under the assumption is:

$$f_k(x) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma_k^2}\right)$$

And for **a linear DA**, we further assumes **equal variances across classes**:

$$\sigma_0^2 = \sigma_1^2 = \cdots = \sigma_{K-1}^2 = \sigma^2$$

This assumption simplifies the decision boundaries to linear functions.

From the previous section, we know that:

$$p_k(x) = \frac{f_k(x) \cdot \pi_k}{\sum_{\ell \in C} \pi_\ell f_\ell(x)}$$

Substitute $f_k(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma^2}\right)$ into the formula:

$$p_k(x) = \frac{\pi_k \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma^2}\right)}{\sum_{\ell \in C} \pi_\ell \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_\ell)^2}{2\sigma^2}\right)} = \frac{\pi_k \exp\left(-\frac{(x-\mu_k)^2}{2\sigma^2}\right)}{\sum_{\ell \in C} \pi_\ell \exp\left(-\frac{(x-\mu_\ell)^2}{2\sigma^2}\right)}$$

And just as before, we assign $X = x$ to the class k that maximizes $p_k(x)$. This simplifies to:

$$\arg\max_{k \in C} p_k(x) = \arg\max_{k \in C} \log\left(p_k(x)\right)$$

Taking the log of the numerator:

$$\log\left(p_k(x)\right) = \log\left(\pi_k \exp\left(-\frac{(x-\mu_k)^2}{2\sigma^2}\right)\right) - \log\left(\sum_{\ell \in C} \pi_\ell \exp\left(-\frac{(x-\mu_\ell)^2}{2\sigma^2}\right)\right)$$

The first term simplifies:

$$\log\left(\pi_k \exp\left(-\frac{(x-\mu_k)^2}{2\sigma^2}\right)\right) = \log\left(\pi_k\right) - \frac{(x-\mu_k)^2}{2\sigma^2}$$

The second term $\log\left(\sum_{\ell \in C} \pi_\ell \exp\left(-\frac{(x-\mu_\ell)^2}{2\sigma^2}\right)\right)$ is constant across all classes and does not affect the classification rule.

The quadratic term $\frac{(x-\mu_k)^2}{2\sigma^2}$ can be expanded:

$$(x-\mu_k)^2 = x^2 - 2x\mu_k + \mu_k^2.$$

Substituting this into $\delta_k(x)$:

$$\delta_k(x) = \log\left(\pi_k\right) - \frac{x^2 - 2x\mu_k + \mu_k^2}{2\sigma^2}.$$

Simplify the terms:

$$\delta_k(x) = \frac{\mu_k}{\sigma^2}x - \frac{\mu_k^2}{2\sigma^2} + \log\left(\pi_k\right)$$

Thus, the logarithm of the posterior simplifies to the discriminant function:

$$\log\left(p_k(x)\right) = \delta_k(x) = \frac{\mu_k}{\sigma^2}x - \frac{\mu_k^2}{2\sigma^2} + \log\left(\pi_k\right)$$

This is linear in x, justifying the name Linear Discriminant Analysis.

Remark.

Linear DA simplifies the process of classifying data by using the assumption of equal variances (σ^2) across classes. This results in discriminant functions $\delta_k(x)$ that are linear equations in x, where the relationship between the features and the classification decision is expressed as a straight line in the case of one feature, $p = 1$. Consequently, LDA does not require more complex or nonlinear decision boundaries; instead, it leverages the properties of Gaussian distributions and the equality of variances to separate classes with simple, interpretable linear functions that depend on the means, priors, and variances of the data.

7.2.1 Application of univariate LDA: binary classification

In the case of two classes $(K = 2)$:

$$\arg \max_{k\in\{0,1\}} \delta_k(x) = \arg \max_{k\in\{0,1\}} \left[\frac{\mu_k}{\sigma^2}x - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)\right].$$

To classify x, compare the discriminant functions for $k = 0$ and $k = 1$:

$$\delta_1(x) \geq \delta_0(x).$$

Substituting the expressions for $\delta_k(x)$:

$$\frac{\mu_1}{\sigma^2}x - \frac{\mu_1^2}{2\sigma^2} + \log(\pi_1) \geq \frac{\mu_0}{\sigma^2}x - \frac{\mu_0^2}{2\sigma^2} + \log(\pi_0)$$

Rearrange the inequality:

$$\left(\frac{\mu_1}{\sigma^2} - \frac{\mu_0}{\sigma^2}\right)x \geq \frac{\mu_1^2}{2\sigma^2} - \frac{\mu_0^2}{2\sigma^2} + \log\left(\frac{\pi_1}{\pi_0}\right)$$

Factorize:

$$x \geq \frac{\frac{\mu_1^2}{2} - \frac{\mu_0^2}{2} + \sigma^2 \log\left(\frac{\pi_1}{\pi_0}\right)}{\mu_1 - \mu_0}$$

Assuming $\pi_0 = \pi_1$, the logarithmic term disappears:

$$x \geq \frac{\mu_0 + \mu_1}{2}$$

The decision boundary is:

$$x = \frac{\mu_0 + \mu_1}{2}$$

And we assign class 1 if $x \geq \frac{\mu_0 - \mu_1}{2}$.

Remark.

For binary classification, the LDA decision boundary is linear and depends on the means and variance of the classes.

Example.

Consider a classification with means being $\mu_0 = -1.5, \mu_1 = 1.5$ and a shared variance $\sigma = 1$.

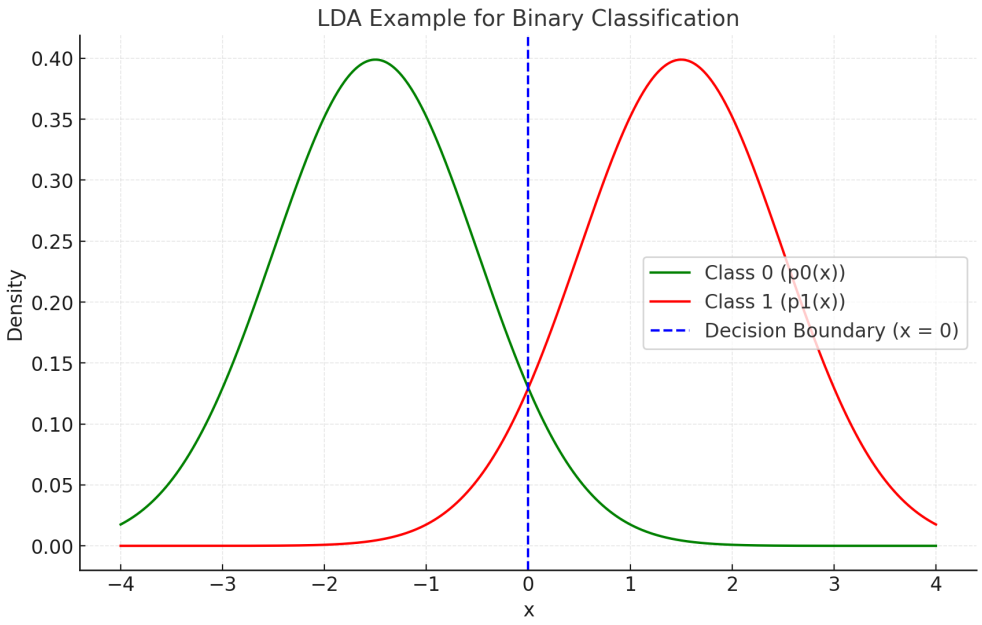

Figure 16: LDA Example for Binary Classification

The vertical dashed line at $x = 0$ is the Bayes decision boundary:

$$x = \frac{\mu_0 + \mu_1}{2} = \frac{-1.5 + 1.5}{2} = 0.$$

Observations with $x < 0$ are classified as class 0 , and those with $x \geq 0$ are classified as class 1.

From the example above, we can conclude some visual perspective:

- If $\pi_0 > \pi_1$, the green curve $(p_0(x))$ will generally have a higher amplitude across x. As a result, we will have a dominance of a class for a range of specific feature values.
- If $\pi_0 < \pi_1$, the red curve $(p_1(x))$ will have a higher amplitude.
- When we have a smaller variance, its curve will be narrower and taller, reflecting that observations are more concentrated around the mean μ_k.
- Conversely, a larger variance spreads the curve, lowering the peak amplitude.
- Low overlap (e.g., means far apart or small variances) means the classifier will have high confidence and accuracy in separating classes.
- High overlap indicates greater uncertainty in classification and a higher likelihood of misclassifications near the decision boundary.

7.2.2 Problem with univariate LDA: estimation under LDA

As we discuess, in order to classify a new observation $X = x$ using the Bayes decision rule based on the discriminant function $\delta_k(x)$.

$$\arg\max_{k \in C} \delta_k(x) = \arg\max_{k \in C} \left(\frac{\mu_k}{\sigma^2} x - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k) \right)$$

In practice however, we often do not know the parameters μ_k, σ^2, π_k precisely. Thus, our solution is to Use training data to estimate these parameters (e.g., using MLE).

Given training data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where x_i are feature values and y_i are class labels:

- **Prior probability:**

$$\hat{\pi}_k = \frac{n_k}{n}$$

where n_k is the number of observations in class k, and n is the total number of observations.

- **Mean:**

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i$$

This is the average of feature values for observations in class k.

- **Variance:**

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2$$

This combines variances within each class into a pooled estimate.

Remark.

The above estimates $(\hat{\pi}_k, \hat{\mu}_k, \hat{\sigma}^2)$ are Maximum Likelihood Estimates under the assumption of normality.

Once the parameters are estimated, we use a "plug-in" approach to compute the discriminant function:

$$\hat{\delta}_k(x) = \frac{\hat{\mu}_k}{\hat{\sigma}^2} x - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k)$$

This is simply the discriminant function $\delta_k(x)$, but with estimated parameters. And we also assign x to the class k that maximizes the plug-in discriminant function:

$$\arg\max_{k \in C} \hat{\delta}_k(x)$$

7.2.3 Proofs for the estimations

Proof for prior probability estimates

We have training data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where $y_i \in \{0, 1, \ldots, K-1\}$ represents the class labels.
Let n_k denote the number of observations belonging to class k:

$$n_k = \sum_{i=1}^{n} 1\{y_i = k\}$$

where $1\{y_i = k\}$ is an indicator function that equals 1 if $y_i = k$ and 0 otherwise.
The goal is to estimate $\pi_k = P(Y = k)$, the prior probability of class k.
The likelihood of observing the class labels $y_1, y_2, \ldots, y_n$ under the assumption of π_k is:

$$L(\pi_0, \pi_1, \ldots, \pi_{K-1}) = \prod_{i=1}^{n} \pi_{y_i}$$

Each observation contributes the prior probability π_k corresponding to its class y_i.
Rewriting using counts n_k:

$$L(\pi_0, \pi_1, \ldots, \pi_{K-1}) = \prod_{k=0}^{K-1} \pi_k^{n_k}$$

Taking the natural logarithm of the likelihood to simplify:

$$\ell(\pi_0, \pi_1, \ldots, \pi_{K-1}) = \log L(\pi_0, \pi_1, \ldots, \pi_{K-1}) = \sum_{k=0}^{K-1} n_k \log \pi_k$$

Since π_k represents probabilities, they must satisfy:

$$\sum_{k=0}^{K-1} \pi_k = 1$$

To incorporate the constraint, we define a Lagrangian:

$$\mathcal{L}(\pi_0, \pi_1, \ldots, \pi_K, \lambda) = \sum_{k=0}^{K} n_k \log(\pi_k) + \lambda \left(1 - \sum_{k=0}^{K} \pi_k\right)$$

Take the derivative of $\mathcal{L}$ with respect to π_k:

$$\frac{\partial \mathcal{L}}{\partial \pi_k} = \frac{n_k}{\pi_k} - \lambda$$

Setting $\frac{\partial \mathcal{L}}{\partial \pi_k} = 0$, we get:

$$\lambda = \frac{n_k}{\pi_k}$$

Since λ is the same for all k, we have:

$$\pi_k = \frac{n_k}{\lambda}$$

Take the derivative of $\mathcal{L}$ with respect to λ to enforce the constraint:

$$\frac{\partial \mathcal{L}}{\partial \lambda} = 1 - \sum_{k=0}^{K} \pi_k = 0 \Longrightarrow \sum_{k=0}^{K} \pi_k = 1$$

Substituting $\pi_k = \frac{n_k}{\lambda}$ into the constraint:

$$\sum_{k=0}^{K} \frac{n_k}{\lambda} = 1 \Longrightarrow \lambda = \sum_{k=0}^{K} n_k = n$$

We have for π_k:

$$\pi_k = \frac{n_k}{n}$$

Proof for mean

For all $k \in \{0, 1, \ldots, K-1\}$:

$$L\left(\mu_0, \ldots, \mu_{K-1}, \sigma^2\right) = \prod_{k=0}^{K-1} \prod_{i:y_i=k} f_k(x_i)$$

Expanding $f_k(x)$:

$$L\left(\mu_0, \ldots, \mu_{K-1}, \sigma^2\right) = \prod_{k=0}^{K-1} \prod_{i:y_i=k} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i - \mu_k)^2}{2\sigma^2}\right)$$

Simplify the likelihood:

$$L\left(\mu_0, \ldots, \mu_{K-1}, \sigma^2\right) = \prod_{k=0}^{K-1} \left[\frac{1}{(2\pi\sigma^2)^{n_k/2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i:y_i=k} (x_i - \mu_k)^2\right)\right]$$

Take the log-likelihood:

$$\ell\left(\mu_0, \ldots, \mu_{K-1}, \sigma^2\right) = \sum_{k=0}^{K-1} \left[-\frac{n_k}{2} \log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2} \sum_{i:y_i=k} (x_i - \mu_k)^2\right]$$

To find the MLE for μ_k, take the partial derivative of the log-likelihood with respect to μ_k:

$$\frac{\partial \ell}{\partial \mu_k} = -\frac{1}{2\sigma^2} \frac{\partial}{\partial \mu_k} \sum_{i:y_i=k} (x_i - \mu_k)^2$$

Expand $(x_i - \mu_k)^2$:

$$\sum_{i:y_i=k} (x_i - \mu_k)^2 = \sum_{i:y_i=k} \left[x_i^2 - 2x_i\mu_k + \mu_k^2\right]$$

Take the derivative:

$$\frac{\partial}{\partial \mu_k} \sum_{i:y_i=k} (x_i - \mu_k)^2 = -2 \sum_{i:y_i=k} x_i + 2n_k\mu_k$$

Simplify:

$$\frac{\partial \ell}{\partial \mu_k} = -\frac{1}{\sigma^2} \left(-\sum_{i:y_i=k} x_i + n_k\mu_k \right)$$

Set $\frac{\partial \ell}{\partial \mu_k} = 0$:

$$\sum_{i:y_i=k} x_i = n_k\mu_k$$

Solve for μ_k:

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i$$

Proof for variance

To find the MLE for σ^2, take the partial derivative of the log-likelihood with respect to σ^2:

$$\frac{\partial \ell}{\partial \sigma^2} = \sum_{k=0}^{K-1} \left[-\frac{n_k}{2\sigma^2} + \frac{1}{2\left(\sigma^2\right)^2} \sum_{i:y_i=k} (x_i - \mu_k)^2 \right]$$

Set $\frac{\partial \ell}{\partial \sigma^2} = 0$:

$$\sum_{k=0}^{K-1} \frac{n_k}{\sigma^2} = \sum_{k=0}^{K-1} \frac{1}{\sigma^4} \sum_{i:y_i=k} (x_i - \mu_k)^2$$

Multiply through by σ^2:

$$\sum_{k=0}^{K-1} n_k = \frac{1}{\sigma^2} \sum_{k=0}^{K-1} \sum_{i:y_i=k} (x_i - \mu_k)^2$$

Simplify:

$$\sigma^2 = \frac{1}{n} \sum_{k=0}^{K-1} \sum_{i:y_i=k} (x_i - \mu_k)^2 .$$

This is the MLE for the variance: the pooled sample variance across all classes.

7.3 LDA with more than one feature

Similarily, for $p > 1$, assuming $X \mid Y = k$ is modeled as a multivariate normal distribution:

$$X \mid Y = k \sim \mathcal{N}_p\left(\mu_k, \Sigma\right)$$

where:

- μ_k is the mean vector of class k (p-dimensional).
- Σ is the shared covariance matrix across all classes $(p \times p)$.

The likelihood $f_k(x)$ of $X = x$ given $Y = k$ is:

$$f_k(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2} \left(x - \mu_k\right)^\top \Sigma^{-1} \left(x - \mu_k\right) \right)$$

The logarithm of $f_k(x)\pi_k$ is:

$$\log\left(f_k(x)\pi_k\right) = \log\left(\pi_k\right) - \frac{1}{2}\log\left((2\pi)^p|\Sigma|\right) - \frac{1}{2}\left(x - \mu_k\right)^\top \Sigma^{-1} \left(x - \mu_k\right)$$

Ignoring constant terms $-\frac{1}{2}\log\left((2\pi)^p|\Sigma|\right)$, which are the same for all classes and do not affect the classification, we simplify to:

$$\log\left(f_k(x)\pi_k\right) = \log\left(\pi_k\right) - \frac{1}{2}\left(x-\mu_k\right)^\top \Sigma^{-1}\left(x-\mu_k\right)$$

Expand the quadratic term $\left(x-\mu_k\right)^\top \Sigma^{-1}\left(x-\mu_k\right)$:

$$\left(x-\mu_k\right)^\top \Sigma^{-1}\left(x-\mu_k\right) = x^\top\Sigma^{-1}x - 2x^\top\Sigma^{-1}\mu_k + \mu_k^\top\Sigma^{-1}\mu_k$$

Substituting this into the discriminant function:

$$\log\left(f_k(x)\pi_k\right) = \log\left(\pi_k\right) - \frac{1}{2}\left(x^\top\Sigma^{-1}x - 2x^\top\Sigma^{-1}\mu_k + \mu_k^\top\Sigma^{-1}\mu_k\right)$$

Rearrange terms:

$$\log\left(f_k(x)\pi_k\right) = -\frac{1}{2}x^\top\Sigma^{-1}x + x^\top\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^\top\Sigma^{-1}\mu_k + \log\left(\pi_k\right)$$

For classification, the term $-\frac{1}{2}x^\top\Sigma^{-1}x$ is the same for all classes, so it can be omitted. The simplified discriminant function becomes:

$$\delta_k(x) = x^\top\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^\top\Sigma^{-1}\mu_k + \log\left(\pi_k\right)$$

Remark.

We can see that for $p=1$:

$$\delta_k(x) = \frac{\mu_k}{\sigma^2}x - \frac{\mu_k^2}{2\sigma^2} + \log\left(\pi_k\right)$$

Remark.

As usual, boundaries are defined by $\delta_k(x) = \delta_\ell(x)$ for $k \neq \ell$, resulting in linear hyperplanes in $\mathbb{R}^p$.

Example.

Consider the following example:

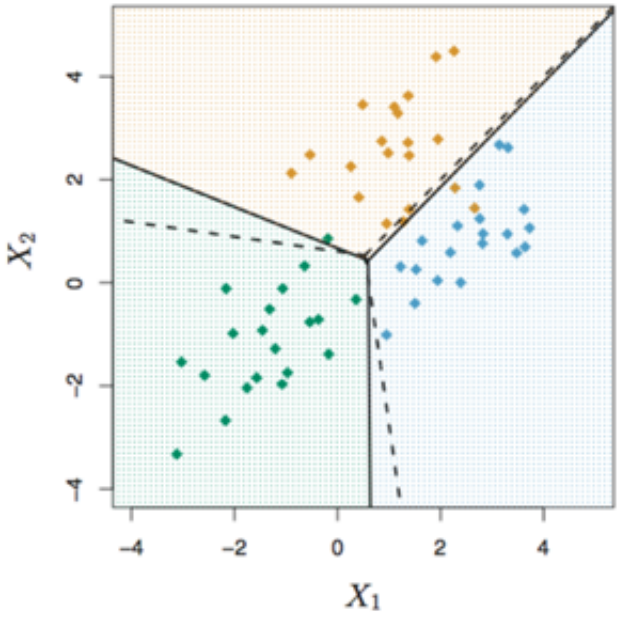

Figure 17: Visualization of multivariate LDA

We have dashed lines show the ideal decision boundaries based on true distributions. And solid lines show the decision boundaries estimated from the data using LDA.

7.3.1 Estimation with multivariate LDA

Given training data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$:

- **Class proportions:**
$$\hat{\pi}_k = \frac{n_k}{n}$$
where n_k is the number of observations in class k.
- **Mean vectors:**
$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i$$
where x_i is a p-dimensional feature vector.
- **Covariance matrix:**
$$\hat{\Sigma} = \frac{1}{n} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^\top$$

Using the estimated parameters $\left(\hat{\pi}_k, \hat{\mu}_k, \hat{\Sigma}\right)$, the discriminant function is:

$$\hat{\delta}_k(x) = x^\top \hat{\Sigma}^{-1} \hat{\mu}_k - \frac{1}{2} \hat{\mu}_k^\top \hat{\Sigma}^{-1} \hat{\mu}_k + \log(\hat{\pi}_k)$$

And we assign x to the class k that maximizes $\hat{\delta}_k(x)$:

$$\arg\max_{k \in C} \hat{\delta}_k(x)$$

As a outcome, LDA results in linear decision boundaries in the feature space $\mathbb{R}^p$, separating classes based on their estimated means and shared covariance structure.

7.4 Logistic regression vs. LDA

7.4.1 Similarities

For binary classification in LDA, the log-odds (logarithm of the ratio of posterior probabilities for the two classes) has the following form:

$$\log\left(\frac{p_1(x)}{1 - p_1(x)}\right) = \log\left(\frac{p_1(x)}{p_0(x)}\right) = c_0 + c_1 x_1 + c_2 x_2 + \cdots + c_p x_p$$

where $c_0, c_1, \ldots, c_p$ are coefficients derived from the prior probabilities π_0, π_1, the class means μ_0, μ_1, and the shared covariance matrix Σ. We will prove below.

In logistic regression, the log-odds are modeled directly as a linear function:

$$\log\left(\frac{p_1(x)}{1 - p_1(x)}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

The log-odds for LDA is a linear function of the features $x_1, x_2, \ldots, x_p$, similar to logistic regression.

Proof. For $p_1(x)$ and $p_0(x)$, we have:

$$p_1(x) = \frac{\pi_1 f_1(x)}{\pi_1 f_1(x) + \pi_0 f_0(x)}, \quad p_0(x) = \frac{\pi_0 f_0(x)}{\pi_1 f_1(x) + \pi_0 f_0(x)}$$

Thus, the ratio becomes:

$$\frac{p_1(x)}{p_0(x)} = \frac{\pi_1 f_1(x)}{\pi_0 f_0(x)}$$

The likelihood $f_k(x)$ for $X \mid Y = k \sim \mathcal{N}_p(\mu_k, \Sigma)$ is:

$$f_k(x) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_k)^\top \Sigma^{-1} (x - \mu_k)\right)$$

Taking the ratio of likelihoods $f_1(x)$ and $f_0(x)$:

$$\frac{f_1(x)}{f_0(x)} = \exp\left(-\frac{1}{2}(x-\mu_1)^\top \Sigma^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_0)^\top \Sigma^{-1}(x-\mu_0)\right)$$

Simplify the exponent:

$$(x-\mu_0)^\top \Sigma^{-1}(x-\mu_0) - (x-\mu_1)^\top \Sigma^{-1}(x-\mu_1) = 2x^\top \Sigma^{-1}(\mu_1-\mu_0) - \mu_1^\top \Sigma^{-1}\mu_1 + \mu_0^\top \Sigma^{-1}\mu_0$$

Thus:

$$\frac{f_1(x)}{f_0(x)} = \exp\left(x^\top \Sigma^{-1}(\mu_1-\mu_0) - \frac{1}{2}\left(\mu_1^\top \Sigma^{-1}\mu_1 - \mu_0^\top \Sigma^{-1}\mu_0\right)\right)$$

Taking the logarithm of $\frac{p_1(x)}{p_0(x)}$:

$$\log\left(\frac{p_1(x)}{p_0(x)}\right) = \log\left(\frac{\pi_1}{\pi_0}\right) + x^\top \Sigma^{-1}(\mu_1-\mu_0) - \frac{1}{2}\left(\mu_1^\top \Sigma^{-1}\mu_1 - \mu_0^\top \Sigma^{-1}\mu_0\right)$$

Let:

- $c_0 = \log\left(\frac{\pi_1}{\pi_0}\right) - \frac{1}{2}\left(\mu_1^\top \Sigma^{-1}\mu_1 - \mu_0^\top \Sigma^{-1}\mu_0\right)$
- $c = \Sigma^{-1}(\mu_1-\mu_0)$, where c is a vector of coefficients

Then:

$$\log\left(\frac{p_1(x)}{p_0(x)}\right) = c_0 + c^\top x = c_0 + c_1x_1 + c_2x_2 + \cdots + c_px_p$$

■

7.4.2 Differences

The differences of the logistic regression and LDA has three major differences:

1. **When it comes to assumptions:**
   - For logistic regression, there is no assumptions about the distribution of X or $P(X \mid Y)$. And only models the conditional probability $P(Y \mid X)$ using a logistic function.
   - For LDA, we assume $X \mid Y = k$ follows a multivariate normal distribution with shared covariance Σ across all classes.
2. **When it comes to paramter estimations:**
   - For logistic regression, we estimate parameters ($\beta_0, \beta_1, \ldots, \beta_p$) by maximizing the conditional likelihood:
     $$\prod_{i=1}^{n} P(Y = y_i \mid X = x_i)$$
     This approach is referred to as discriminative learning.
   - For LDA, we estimate parameters (π_k, μ_k, Σ) by maximizing the joint likelihood:
     $$\prod_{i=1}^{n} P(X = x_i, Y = y_i)$$
     This approach is called generative learning.
3. In terms of performance, logistic regression is more robust when classes are not well-separated. This is because logistic regression does not assume the classes being separated well. On the other hand, LDA performs better when the normality and equal covariance assumptions hold, and the classes are well-separated. This is because LDA assumes to have a linear boundary. And when the classes are not well-separated, the probability distribution becomes overlapping, and the probability becomes similar resulting in ambiguity.

7.5 Other forms of DA: quadratic DA

There are other forms of DA with a much more relaxed assumptions on the covariance. Unlike LDA, which assumes a shared covariance matrix (Σ) across all classes, QDA allows the covariance matrix to vary by class. We still assume $X \mid Y = k \sim \mathcal{N}(\mu_k, \Sigma_k)$, where Σ_k is the covariance matrix specific to class k.

This flexibility allows QDA to capture more complex class distributions at the cost of estimating more parameters. The decision boundaries are quadratic instead of linear.

7.5.1 Quadratic DA with single feature

By assumption, we know for $X \mid Y = k \sim \mathcal{N}(\mu_k, \sigma_k^2)$, the likelihood is:

$$f_k(x) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma_k^2}\right)$$

Using Bayes' theorem, the posterior probability $p_k(x)$ becomes:

$$p_k(x) = \frac{\pi_k f_k(x)}{\sum_{\ell \in C} \pi_\ell f_\ell(x)} = \frac{\pi_k \frac{1}{\sigma_k} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma_k^2}\right)}{\sum_{\ell \in C} \pi_\ell \frac{1}{\sigma_\ell} \exp\left(-\frac{(x-\mu_\ell)^2}{2\sigma_\ell^2}\right)}$$

As usual, we classify $X = x$ to the class k that maximizes $\log(p_k(x))$:

$$\arg\max_{k \in C} \log(p_k(x))$$

Taking the log to simplify the computations:

$$\log(\pi_k f_k(x)) = \log(\pi_k) + \log\left(\frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma_k^2}\right)\right)$$

Expanding the log terms:

$$\log(\pi_k f_k(x)) = \log(\pi_k) - \frac{1}{2}\log\left(2\pi\sigma_k^2\right) - \frac{(x-\mu_k)^2}{2\sigma_k^2}$$

- For constant term:

  $$-\frac{1}{2}\log\left(2\pi\sigma_k^2\right)$$

  are independent of x and do not affect the maximization directly.

- For Quadratic term:

  $$-\frac{(x-\mu_k)^2}{2\sigma_k^2}$$

  Expanding $(x-\mu_k)^2$:

  $$(x-\mu_k)^2 = x^2 - 2x\mu_k + \mu_k^2$$

  Substituting back:

  $$-\frac{(x-\mu_k)^2}{2\sigma_k^2} = -\frac{x^2}{2\sigma_k^2} + \frac{\mu_k}{\sigma_k^2}x - \frac{\mu_k^2}{2\sigma_k^2}$$

Combining all terms, the discriminant function for QDA in the univariate case is:

$$\delta_k(x) = -\frac{x^2}{2\sigma_k^2} + \frac{\mu_k}{\sigma_k^2}x - \frac{\mu_k^2}{2\sigma_k^2} + \log(\pi_k) - \frac{1}{2}\log\left(2\pi\sigma_k^2\right)$$

Remark.

This is quadratic in x, giving QDA its name.

7.5.2 Quadratic DA with more than one features

We assume for $X \mid Y = k \sim \mathcal{N}_p(\mu_k, \Sigma_k)$, the likelihood is:

$$f_k(x) = \frac{1}{(2\pi)^{p/2} |\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)^\top \Sigma_k^{-1}(x-\mu_k)\right)$$

And similarly, we take $\arg\max_k P(Y = k \mid X = x) = \arg\max_k \pi_k f_k(x)$.

Taking the logarithm to simplify computation:

$$\log(\pi_k f_k(x)) = \log(\pi_k) + \log\left(\frac{1}{(2\pi)^{p/2} |\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)^\top \Sigma_k^{-1}(x-\mu_k)\right)\right)$$

Expanding the terms:

$$\log(\pi_k f_k(x)) = \log(\pi_k) - \frac{1}{2}\log((2\pi)^p |\Sigma_k|) - \frac{1}{2}(x-\mu_k)^\top \Sigma_k^{-1}(x-\mu_k)$$

Expanding the quadratic term $(x-\mu_k)^\top \Sigma_k^{-1}(x-\mu_k)$:

$$(x-\mu_k)^\top \Sigma_k^{-1}(x-\mu_k) = x^\top \Sigma_k^{-1} x - 2x^\top \Sigma_k^{-1}\mu_k + \mu_k^\top \Sigma_k^{-1}\mu_k$$

Substituting this back into the log expression:

$$\log(\pi_k f_k(x)) = \log(\pi_k) - \frac{1}{2}\log((2\pi)^p |\Sigma_k|) - \frac{1}{2}\left(x^\top \Sigma_k^{-1} x - 2x^\top \Sigma_k^{-1}\mu_k + \mu_k^\top \Sigma_k^{-1}\mu_k\right)$$

Reorganizing the terms, the discriminant function $\delta_k(x)$ for QDA is:

$$\delta_k(x) = -\frac{1}{2}x^\top \Sigma_k^{-1} x + x^\top \Sigma_k^{-1}\mu_k - \frac{1}{2}\mu_k^\top \Sigma_k^{-1}\mu_k + \log(\pi_k) - \frac{1}{2}\log(|\Sigma_k|)$$

And as usual, the decision boundary between classes k and ℓ is defined by the set of x where:

$$\delta_k(x) = \delta_\ell(x)$$

This boundary is quadratic in x, allowing QDA to separate classes with more complex boundaries than LDA.

7.5.3 Estimations in QDA

Given training data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where x_i are feature vectors (p-dimensional), and y_i are class labels ($k \in \{1, 2, \ldots, K\}$).

- **Prior probability:**
$$\hat{\pi}_k = \frac{n_k}{n}$$
- **Mean Vectors** (μ_k)**:**
$$\hat{\mu}_k = \frac{1}{n_k}\sum_{i:y_i=k} x_i,$$
- **Covariance Matrices** (Σ_k)**:**
$$\hat{\Sigma}_k = \frac{1}{n_k}\sum_{i:y_i=k}(x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^\top,$$

Once the parameters ($\hat{\pi}_k, \hat{\mu}_k, \hat{\Sigma}_k$) are estimated, they are substituted into the QDA discriminant function:

$$\hat{\delta}_k(x) = -\frac{1}{2}x^\top \hat{\Sigma}_k^{-1} x + x^\top \hat{\Sigma}_k^{-1}\hat{\mu}_k - \frac{1}{2}\hat{\mu}_k^\top \hat{\Sigma}_k^{-1}\hat{\mu}_k + \log(\hat{\pi}_k) - \frac{1}{2}\log\left(\left|\hat{\Sigma}_k\right|\right).$$

7.6 Issues with QDA and LDA in high dimensions

For LDA, we assumes a shared covariance matrix (Σ) across all classes. And the total number of parameters is:

$$(K-1) + pK + \frac{p(p+1)}{2}$$

- $(K-1)$: Prior probabilities (π_k)
- pK: Class means (μ_k)
- $\frac{p(p+1)}{2}$: Parameters for the shared covariance matrix (Σ)

For QDA, we allow a separate covariance matrix (Σ_k) for each class. Making the total number of parameters to be:

$$(K-1) + pK + \frac{p(p+1)}{2}K$$

When p is large compared to n, the number of parameters becomes excessive, leading to:

- Overfitting
- High variance in parameter estimates
- Poor generalization

Thus, LDA and QDA are not well-suited for high-dimensional data without dimensionality reduction techniques or regularization.

7.7 Non-parametric approach to classification: Naive Bayes

Naive bayes assume the features $X_1, X_2, \ldots, X_p$ are conditionally independent given the class Y:

$$f_k(x) = \prod_{j=1}^{p} f_{k,j}\left(x_j\right)$$

where $f_{k,j}\left(x_j\right)$ is the marginal density of feature X_j for class k.

This assumption simplifies the model, as it eliminates the need to estimate covariances between features. Making it handle both quantitative (numerical) and categorical features easily. Despite its strong independence assumption, Naive Bayes often performs surprisingly well in practice.

Each feature $X_j \mid Y = k$ is modeled as:

$$X_j \mid Y = k \sim \mathcal{N}\left(\mu_{k,j}, \sigma_{k,j}^2\right)$$

where:

- $\mu_{k,j}$: Mean of feature j in class k
- $\sigma_{k,j}^2$: Variance of feature j in class k

The conditional likelihood is:

$$f_k(x) = \prod_{j=1}^{p} \frac{1}{\sqrt{2\pi\sigma_{k,j}^2}} \exp\left(-\frac{(x_j - \mu_{k,j})^2}{2\sigma_{k,j}^2}\right)$$

To classify $X = x$, we only care about the class that maximizes the posterior probability. The denominator $\sum_{\ell \in C} \pi_\ell \prod_{j=1}^{p} f_{\ell,j}\left(x_j\right)$ is constant for all classes and can be ignored:

$$\arg\max_k P(Y = k \mid X = x) = \arg\max_k \pi_k \prod_{j=1}^{p} f_{k,j}\left(x_j\right)$$

Taking the logarithm simplifies the computation:

$$\log\left(\pi_k \prod_{j=1}^{p} f_{k,j}\left(x_j\right)\right) = \log\left(\pi_k\right) + \sum_{j=1}^{p} \log\left(f_{k,j}\left(x_j\right)\right)$$

Thus, the discriminant function $\delta_k(x)$ for Naive Bayes is:

$$\delta_k(x) = \log\left(\pi_k\right) + \sum_{j=1}^{p} \log\left(f_{k,j}\left(x_j\right)\right)$$

Substituting into $\log\left(f_{k,j}\left(x_j\right)\right)$:

$$\log\left(f_{k,j}\left(x_j\right)\right) = \log\left(\frac{1}{\sqrt{2\pi\sigma_{k,j}^2}}\right) - \frac{\left(x_j - \mu_{k,j}\right)^2}{2\sigma_{k,j}^2}$$

Expanding the terms:

$$\log\left(f_{k,j}\left(x_j\right)\right) = -\frac{1}{2}\log(2\pi) - \frac{1}{2}\log\left(\sigma_{k,j}^2\right) - \frac{\left(x_j - \mu_{k,j}\right)^2}{2\sigma_{k,j}^2}$$

Summing over all features $j = 1, \ldots, p$:

$$\delta_k(x) = \log\left(\pi_k\right) + \sum_{j=1}^{p}\left(-\frac{1}{2}\log(2\pi) - \frac{1}{2}\log\left(\sigma_{k,j}^2\right) - \frac{\left(x_j - \mu_{k,j}\right)^2}{2\sigma_{k,j}^2}\right)$$

Grouping terms:

$$\delta_k(x) = -\frac{1}{2}\sum_{j=1}^{p}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{p}\log\left(\sigma_{k,j}^2\right) - \frac{1}{2}\sum_{j=1}^{p}\frac{\left(x_j - \mu_{k,j}\right)^2}{\sigma_{k,j}^2} + \log\left(\pi_k\right)$$

Since the constant term $-\frac{1}{2}\sum_{j=1}^{p}\log(2\pi)$ is independent of k, it can be omitted from the classification rule. The final discriminant function is:

$$\delta_k(x) = -\frac{1}{2}\sum_{j=1}^{p}\frac{\left(x_j - \mu_{k,j}\right)^2}{\sigma_{k,j}^2} - \frac{1}{2}\sum_{j=1}^{p}\log\left(\sigma_{k,j}^2\right) + \log\left(\pi_k\right)$$

8 Support vector machine

8.1 Decision boundary in binary classification for logistic regression and LDA

Consider in logistic regression, we have log-odds ratio:

$$\log \frac{P(Y=1 \mid X=x)}{P(Y=0 \mid X=x)} = \beta_0 + \beta^\top x$$

Hence, $P(Y=1 \mid X=\mathbf{x}) \geq P(Y=0 \mid X=\mathbf{x})$ if and only if

$$\beta_0 + \boldsymbol{\beta}^\top \mathbf{x} \geq 0$$

Thus, we know the decision boundary is

$$\left\{\mathbf{x} \in \mathbb{R}^p : \beta_0 + \boldsymbol{\beta}^\top \mathbf{x} = 0\right\}$$

In linear DA, we proven that the LDA uses the discriminant function:

$$\delta_k(x) = x^\top \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^\top \Sigma^{-1} \mu_k + \log \pi_k$$

Hence, $\delta_1(\mathbf{x}) \geq \delta_0(\mathbf{x})$ if and only if

$$\left(\mathbf{x} - \frac{u_0 + u_1}{2}\right)^\top \Sigma^{-1} (u_1 - u_0) + \log \frac{\pi_1}{\pi_0} \geq 0 \quad \text{Proof below}$$

The decision boundary is again linear:

$$\left\{x \in \mathbb{R}^p : \alpha_0 + \alpha^\top x = 0\right\} \quad \text{Proof below}$$

α_0 and α are derived from LDA parameters.

Proof. The LDA classifier assigns $\mathbf{x}$ to class 1 if:

$$\delta_1(\mathbf{x}) \geq \delta_0(\mathbf{x})$$

Substituting for $\delta_1(\mathbf{x})$ and $\delta_0(\mathbf{x})$:

$$\mathbf{x}^\top \Sigma^{-1} \mu_1 - \frac{1}{2} \mu_1^\top \Sigma^{-1} \mu_1 + \log \pi_1 \geq \mathbf{x}^\top \Sigma^{-1} \mu_0 - \frac{1}{2} \mu_0^\top \Sigma^{-1} \mu_0 + \log \pi_0$$

Bring all terms involving $\mathbf{x}$ and constants to one side:

$$\mathbf{x}^\top \Sigma^{-1} \mu_1 - \mathbf{x}^\top \Sigma^{-1} \mu_0 \geq \frac{1}{2} \mu_1^\top \Sigma^{-1} \mu_1 - \frac{1}{2} \mu_0^\top \Sigma^{-1} \mu_0 + \log \pi_0 - \log \pi_1$$

Combine terms involving $\mathbf{x}$:

$$\mathbf{x}^\top \Sigma^{-1} (\mu_1 - \mu_0) \geq \frac{1}{2} \left(\mu_1^\top \Sigma^{-1} \mu_1 - \mu_0^\top \Sigma^{-1} \mu_0\right) + \log \frac{\pi_0}{\pi_1}$$

Rewriting the constant term:

$$\frac{1}{2} \left(\mu_1^\top \Sigma^{-1} \mu_1 - \mu_0^\top \Sigma^{-1} \mu_0\right) + \log \frac{\pi_0}{\pi_1} = -\frac{1}{2} (\mu_0 + \mu_1)^\top \Sigma^{-1} (\mu_1 - \mu_0) + \log \frac{\pi_0}{\pi_1}$$

Thus:

$$\mathbf{x}^\top \Sigma^{-1} (\mu_1 - \mu_0) \geq -\frac{1}{2} (\mu_0 + \mu_1)^\top \Sigma^{-1} (\mu_1 - \mu_0) + \log \frac{\pi_0}{\pi_1}$$

Move the term involving $\frac{\mu_0 + \mu_1}{2}$ to the left-hand side:

$$\left(\mathbf{x} - \frac{\mu_0 + \mu_1}{2}\right)^\top \Sigma^{-1} (\mu_1 - \mu_0) \geq \log \frac{\pi_0}{\pi_1}$$

Finally, reversing the inequality sign by flipping the log ratio:

$$\left(\mathbf{x} - \frac{\mu_0 + \mu_1}{2}\right)^\top \Sigma^{-1} (\mu_1 - \mu_0) + \log \frac{\pi_1}{\pi_0} \geq 0$$

■

Proof. The decision boundary is defined where the two discriminant functions are equal, i.e., where:

$$x^\top \Sigma^{-1} (\mu_1 - \mu_0) = \frac{1}{2} \left(\mu_1^\top \Sigma^{-1} \mu_1 - \mu_0^\top \Sigma^{-1} \mu_0 \right) + \log \frac{\pi_1}{\pi_0}.$$

Take $\alpha = \Sigma^{-1} (\mu_1 - \mu_0)$ and $\alpha_0 = -\frac{1}{2} \left(\mu_1^\top \Sigma^{-1} \mu_1 - \mu_0^\top \Sigma^{-1} \mu_0 \right) - \log \frac{\pi_1}{\pi_0}$. Then the decision boundary equation becomes:

$$x^\top \alpha + \alpha_0 = 0$$

Because $x^\top \alpha = \alpha^\top x$, we have the decision boundary written as:

$$\left\{ x \in \mathbb{R}^p : \alpha_0 + \alpha^\top x = 0 \right\}$$

■

8.2 General set up for decision boundary in binary classification

In this context, the target variable is $y \in \{-1, +1\}$, which is changed from $\{0, 1\}$ for convenience. And consider linear decision boundary:

$$w^\top x + b = 0$$

- $w \in \mathbb{R}^p$: Weight vector defining the direction of the boundary.
- $b \in \mathbb{R}$: Bias term shifting the hyperplane.

A "good" boundary ensures:

$$w^\top x + b > 0, \text{ if } y = 1$$
$$w^\top x + b < 0, \text{ if } y = -1$$

And the classifier assigns points to the correct side of the hyperplane:

$$\text{sign} \left(w^\top x_i + b \right) = \begin{cases} +1, & \text{if } w^\top \mathbf{x}_i + b > 0 \text{ and } y_i = +1 \\ -1, & \text{if } w^\top \mathbf{x}_i + b < 0 \text{ and } y_i = -1 \end{cases} = y_i \text{ for all } i \in \{1, \ldots, n\}$$

8.2.1 Geometric intuition

In a 2-dimensional space $(p = 2)$, the decision boundary is a line. And in higher dimensions $(p > 2)$, the decision boundary becomes a $(p-1)$-dimensional hyperplane.

They divide the feature space into two havles:

- One side corresponds to points classified as $y = +1$.
- The other side corresponds to points classified as $y = -1$.

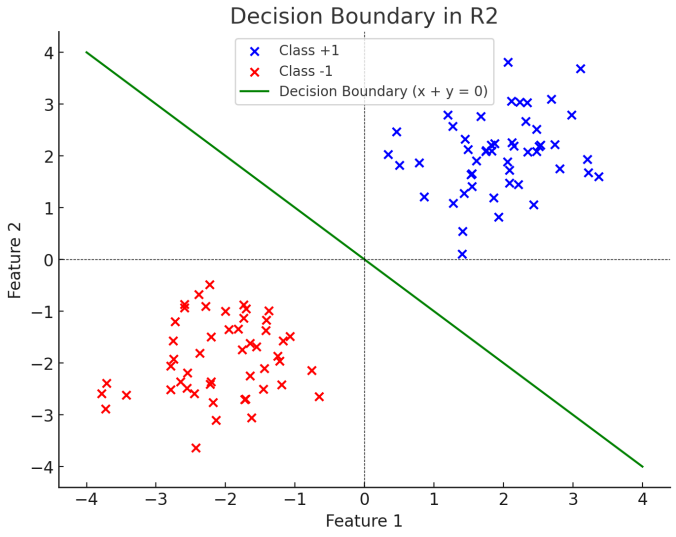

Figure 18: Decision Boundary In R2

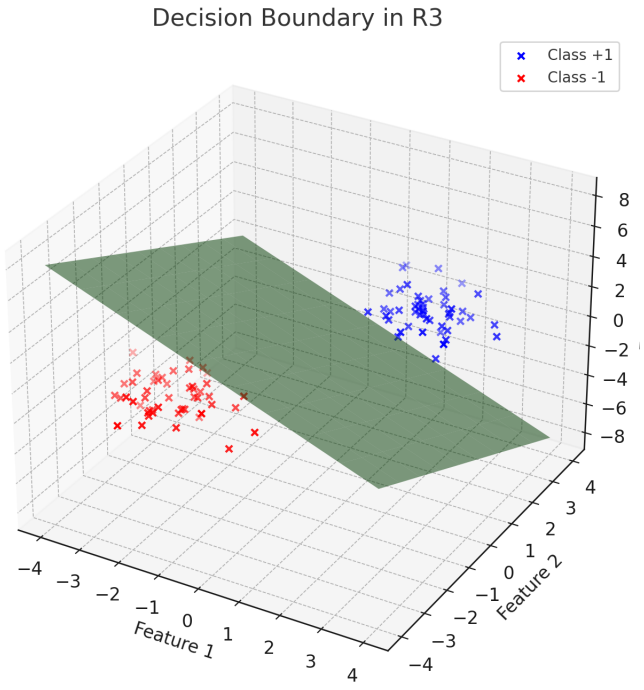

Figure 19: Decision boundary in R3

8.2.2 Challenges in decision boundary and potential solution

When the data is separable, there are many possible hyperplanes that satisfy the condition. Which one should we choose? If no hyperplane exists that perfectly separates the data, the approach is infeasible.

To address these issues, we aim to find the **optimal separating hyperplane**, which maximizes the margin, which means the distance to the closest points of either class.

8.3 Optimal separating hyperplane

8.3.1 Identifying margin of the hyperplane and its geometry

Definition 8.1: Margin

The margin is the perpendicular distance from the hyperplane to the closest point(s) of either class.

Definition 8.2: Optimal separating hyperplane

The optimal separating hyperplane is a hyperplane that:

1. Separates the two classes (no points from either class cross the boundary, assuming separable data).
2. Maximizes the margin, which is the distance between the hyperplane and the closest points from either class.

Remark.

The goal of maximizing the margin is to to ensure that the decision boundary generalizes well to unseen data by maximizing this margin. A classifier with a larger margin is less sensitive to small changes in the data, making it more robust to overfitting.

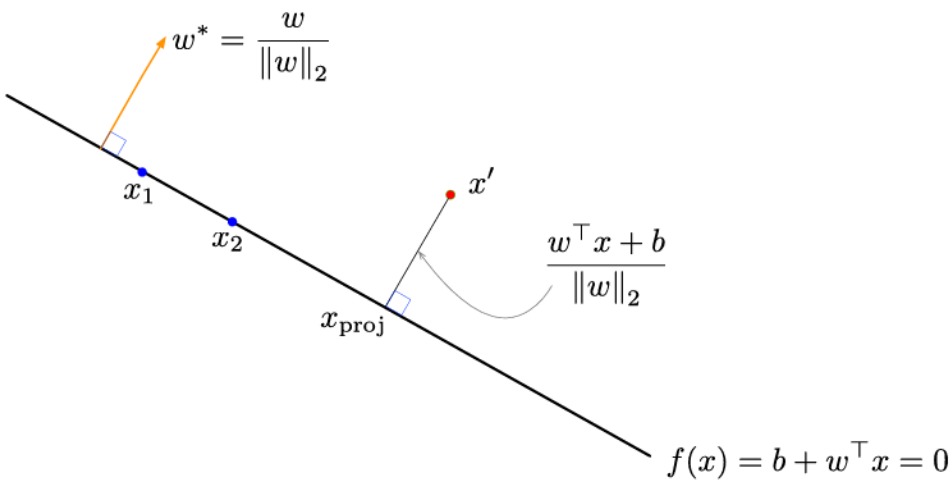

Figure 20: Geometry of points and planes

- The decision hyperplane is orthogonal (perpendicular) to the weight vector w. The weight vector can be normalized:

$$w^* = \frac{w}{\|w\|_2}$$

  w^* is a unit vector pointing in the same direction as w.

- For any two points x_1 and x_2 on the hyperplane, the following holds:

$$w^\top (x_1 - x_2) = 0$$

Proposition 8.3

The perpendicular distance from a point x' to the hyperplane $\left\{x : w^\top x + b = 0\right\}$ is given by:

$$\text{Distance} = \frac{\left|w^\top x' + b\right|}{\|w\|_2}$$

Proof. Let x_{proj} be the projection of x' onto the hyperplane. x_{proj} satisfies the hyperplane equation:

$$w^\top x_{\text{proj}} + b = 0$$

The vector $x' - x_{\text{proj}}$ is parallel to the direction of w. Normalize w to find the direction vector:

$$w^* = \frac{w}{\|w\|_2}$$

w^* is the unit vector in the same direction as w.
The Euclidean distance between x' and x_{proj} is:

$$\|x' - x_{\text{proj}}\|_2$$

This can be rewritten using the projection formula:

$$\|x' - x_{\text{proj}}\|_2 = \frac{\left|w^\top x' + b\right|}{\|w\|_2}$$

■

Now, consider two parallel hyperplanes are defined:

$$w^\top x + b = 1 \quad \text{and} \quad w^\top x + b = -1$$

For a point on the $w^\top x + b = 1$, the perpendicular distance to the decision boundary $w^\top x + b = 0$ is:

$$\text{Distance} = \frac{|1|}{\|w\|_2}$$

Similarly, for a point on the $w^\top x + b = -1$, the perpendicular distance to the decision boundary is also:

$$\frac{|1|}{\|w\|_2}$$

Thus, the margin width is the total distance between these two hyperplanes:

$$\text{Margin Width } = \frac{1}{\|w\|_2} + \frac{1}{\|w\|_2} = \frac{2}{\|w\|_2}$$

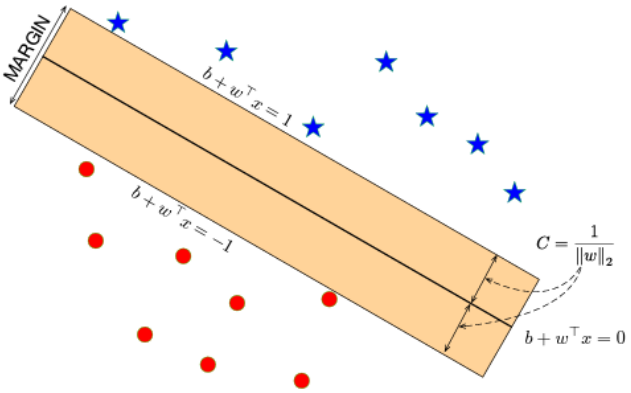

Figure 21: Margin of the hyperplane

8.3.2 Introduction to Hard margin SVM

To ensure all points are correctly classified and prevent them from falling inside the margin, the following constraints are imposed:

$$w^\top x_i + b \geq M \text{ if } y_i = 1$$
$$w^\top x_i + b \leq -M \text{ if } y_i = -1$$

These are unified as:

$$y_i \left(w^\top x_i + b \right) \geq M \quad \forall i \in \{1, \ldots, n\}$$

M represents the margin's scale, and this is called the margin constraint.

There might exist multiple (w, b) satisfy the margin constraints. We want to pick the one that maximizes the width of the margin,

$$\frac{\left| \mathbf{x}^\top w + b \right|}{\|w\|_2} = \frac{M}{\|w\|_2}$$

And it is equivalent to minimize $\|w\|_2$ subject to the margin constraints:

$$\min_{w,b} \frac{\|w\|_2^2}{M^2} \text{ s.t. } y_i \left(w^\top x_i + b \right) \geq M$$

Proposition 8.4

WLOG, we can take $M = 1$.

Proof. If $M \neq 1$, we can rewrite the constraints:

$$y_i \left(w^\top x_i + b\right) \geq M \quad \Longrightarrow \quad y_i \left(\frac{w}{M}^\top x_i + \frac{b}{M}\right) \geq 1$$

By dividing w and b by M, the problem is reformulated with $M = 1$:

$$\min_{\frac{w}{M}, \frac{b}{M}} \frac{\|\frac{w}{M}\|_2^2}{\frac{M}{M}^2} \text{ s.t. } y_i \left(\frac{w}{M}^\top x_i + \frac{b}{M}\right) \geq 1$$

This transformation does not alter the geometry of the hyperplane or the optimization results. It merely standardizes the margin to 1. ■

Thus, the optimization question becomes

$$\min_{w,b} \|w\|_2^2 \text{ s.t. } y_i \left(w^\top x_i + b\right) \geq 1$$

- Points are correctly classified, meaning $y_i \left(w^\top x_i + b\right) \geq 1$.
- The margin is maximized and $\|w\|_2$ minimized.

Definition 8.5: Support vectors

Points that exactly satisfy $y_i \left(w^\top x_i + b\right) = 1$ lie on the margin

Remark.

These support vectors define the optimal hyperplane. Other points farther away from the hyperplane do not influence the solution, which can be shown formally using Karush-Kuhn-Tucker conditions.

Remark.

Hence, this algorithm is called the (hard-margin) Support Vector Machine (SVM). SVM-like algorithms are often called max-margin or large-margin.

8.3.3 Computing hard margin SVM

In practice, SVMs are often solved using the dual formulation, which transforms the problem into a different space where optimization is easier. The dual formulation Leverage the dot product between data points. And it Leads to the introduction of kernel functions in later SVM extensions.

Both concepts are not necessary for the course.

8.4 Soft margin SVM

Real-world datasets are often non-separable, meaning no hyperplane can perfectly classify all points. For example, they can have a lot of overlapping classes.

To handle non-separable data, the hard-margin SVM must be extended to allow misclassifications for some points and flexibility in enforcing the margin constraints.

For soft margin SVM, we include the slack variable $\zeta = (\zeta_1, \ldots, \zeta_n)$ in the optimization problem:

$$\min_{w,b,\zeta} \|w\|_2^2 \text{ s.t. } y_i \left(w^\top x_i + b\right) \geq 1 - \zeta_i, \quad \zeta_i \geq 0, \quad \forall i \text{ and } \sum_{i=1}^{n} \zeta_i \leq K$$

- If $\zeta_i = 0$, the point satisfies the margin constraint $y_i \left(w^\top x_i + b\right) \geq 1$.
- If $0 < \zeta_i \leq 1$, the point is within the margin but correctly classified.
- If $\zeta_i > 1$, the point is misclassified.
- $K \geq 0$ is a hyperparameter controlling the total margin violations allowed.

Remark.

Slack variables (ζ_i) measure the extent to which individual data points violate the margin constraints.

Remark.

A smaller K enforces stricter adherence to the margin, resulting in fewer violations but potentially overfitting the data. A larger K allows more violations, increasing the model's flexibility to handle noisy or overlapping data.

Remark.

$K = 0$ reduces to the hard-margin SVM.

8.4.1 A equivalent formulation

Soft-margin SVM is equivalent to, for some $C = C(K)$,

$$\min_{w,b,\zeta} \|w\|_2^2 + C\sum_{i=1}^{n} \zeta_i \text{ s.t. } y_i\left(w^\top x_i + b\right) \geq 1 - \zeta_i, \quad \zeta_i \geq 0, \quad \forall i$$

The constraint $y_i\left(w^\top x_i + b\right) \geq 1 - \zeta_i$ implies:

$$\zeta_i \geq \max\left(0, 1 - y_i\left(w^\top x_i + b\right)\right)$$

- When $y_i\left(w^\top x_i + b\right) \geq 1, \zeta_i = 0$ (no margin violation).
- When $y_i\left(w^\top x_i + b\right) < 1, \zeta_i = 1 - y_i\left(w^\top x_i + b\right)$ (inside margin or misclassified).

Thus, the slack variables ζ_i can be replaced with the hinge loss:

$$\max\left(0, 1 - y_i\left(w^\top x_i + b\right)\right)$$

Replace $\sum_{i=1}^n \zeta_i$ in the original objective with $\sum_{i=1}^n \max\left(0, 1 - y_i\left(w^\top x_i + b\right)\right)$:

$$\min_{w,b} \|w\|_2^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i\left(w^\top x_i + b\right)\right)$$

Divide the second term by n to express the loss as an average over the data points:

$$\min_{w,b} \|w\|_2^2 + \frac{C}{n}\sum_{i=1}^{n} \max\left(0, 1 - y_i\left(w^\top x_i + b\right)\right)$$

Introduce $\lambda = \frac{1}{nC}$ to rewrite the regularization term:

$$\min_{w,b} \frac{1}{n}\sum_{i=1}^{n} \max\left(0, 1 - y_i\left(w^\top x_i + b\right)\right) + \lambda\|w\|_2^2$$

Thus, we have equivalence:

$$\min_{w,b} \frac{1}{n}\sum_{i=1}^{n} \underbrace{\max\left\{0, 1 - y_i\left(w^\top x_i + b\right)\right\}}_{\text{hinge loss}} + \lambda\|w\|_2^2$$

with $\lambda = \frac{1}{nC}$.

Remark.

Both formulations represent the same trade-off between maximizing the margin $\|w\|_2^2$ and minimizing classification error (hinge loss).

8.4.2 More on hinge loss

We know that the hinge loss function for a single point is:

$$L_{\text{hinge}}\left(w,b\right)=\max\left(0,1-y_i\left(w^\top x_i+b\right)\right)$$

And as before:

- If $y_i\left(w^\top x_i+b\right)\geq 1$:
$$L_{\text{hinge}}=0$$
- If $0<y_i\left(w^\top x_i+b\right)<1$:
$$L_{\text{hinge}}=1-y_i\left(w^\top x_i+b\right)>0$$
- If $y_i\left(w^\top x_i+b\right)<0$:
$$L_{\text{hinge}}=1-y_i\left(w^\top x_i+b\right)>0$$

On the contrary, comparing to $0-1$ loss:

$$L_{0-1}(w,b)=\begin{cases}1 & \text{if } y_i\left(w^\top x_i+b\right)<0\\ 0 & \text{otherwise}\end{cases}=1\left\{y_i\left(w^\top x_i+b\right)<0\right\}$$

We have:

- Hinge loss penalizes points within the margin and misclassified points.
- $0-1$ loss penalizes only misclassified points.

8.5 Limitations of SVM

There are several limitations for SVM:

- SVM predicts class labes without estimating the posterior probabilities $P(Y\mid X)$. On the other hand, logistic regression or other probabilistic extensions of SVM are needed for such outputs.
- If we have multiple class classification problem, SVM is not suitable since it is designed inherently for binary classification. And the multi-class extension of SVM is not straightforward. We may beed to use ad-hoc method, which is not a part of the course.
- The performance of SVM is sensitive to kernel.

8.6 SVM, LDA, and logistic regression

- All three methods create linear decision boundaries.
- LDA and logistic regression privide posterior probabilities while SVM does not directly estimate probabilities.
- SVM performs well for separable and non-separable data. LDA performs well under Gaussian assumptions and separable data. And logistic regression works well for non-separable data.
- LDA and logistic regression naturally extend to multi-class classification. While SVM does not naturally extrend to multi-class classification.

9 Decision tree

9.1 Introduction to decision tree

Definition 9.1: Decision tree

Decision Trees (DTs) is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Tree-based methods are versatile and can be used for both regression (predicting a continuous outcome) and classification (predicting a categorical outcome) tasks.

- One of the biggest advantages of tree-based methods is their interpretability. The tree-like structure provides a clear and intuitive visualization of the decision-making process.
- While interpretable, individual decision trees may not always achieve the highest predictive accuracy compared to more complex models.

To improve predictive performance, multiple decision trees can be combined using techniques like bagging, random forests, and boosting, which are likely discussed later in the presentation.

Example.

Consider the following intuitive example:

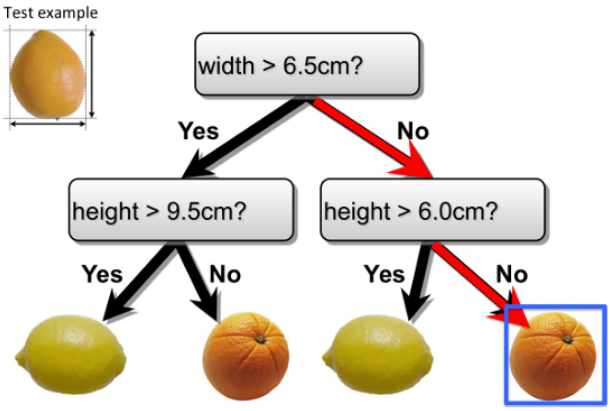

Figure 22: A very simple example of decision tree

The tree makes decisions by splitting the data based on features. In this example, the features are 'width' and 'height'. Each node in the tree represents a decision rule based on a feature. For instance, the top node checks if the 'width' is greater than 6.5cm.
The branches emanating from an node correspond to the possible outcomes of the decision rule (e.g., 'Yes' or 'No'). Following the branches based on an object's features leads to a final node, which provides the final prediction.

Remark.

Discrete features have distinct, separate values (e.g., color: red, green, blue), while continuous features can take on any value within a range (e.g., height, weight).

Remark.

For continuous features, the tree partitions the data by checking if the feature value is greater than or less than a certain threshold. For instance, it might split based on whether 'height' is greater than 9.5cm. These threshold-based splits on continuous features result in rectangular decision boundaries in the feature space:

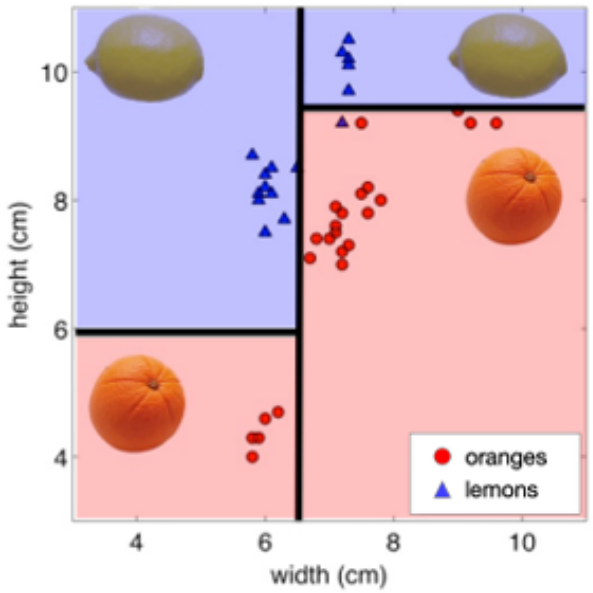

Figure 23: Decision boundary for continuous features in DT

Remark.

The tree splits the data based on the different discrete values of the discrete features. For example, the first split is based on the 'Patrons' feature (number of people in the restaurant), which can have values like 'None', 'Some', and 'Full'.

9.1.1 Some Terminologies

Definition 9.2: Internal node

A node within the tree that represents a feature used for splitting the data.

Definition 9.3: Branch

A connection between nodes that represents the outcome of a split (e.g., 'Yes' or 'No').

Definition 9.4: Children

The nodes that result from splitting a parent node.

Definition 9.5: Leaf node

A terminal node in the tree that provides the final prediction.

9.1.2 In the context of regression and classification

In a decision tree, each path from the root node of the tree to a leaf node defines a distinct region in the feature space.

Take J be the total number of leaf nodes (and hence regions), and R_j be the region defined by the path from the root to the j-th leaf node. Consider $\left\{\left(x^{(j_1)}, y^{(j_1)}\right), \ldots, \left(x^{(j_k)}, y^{(j_k)}\right)\right\}$ be the training data points that fall into region R_j. Then, the prediction for a new data point falling into region R_j is:

- **Regression:** $y^{R_j} = \frac{1}{k}\sum_{i=1}^{k} y^{(j_i)}$ (the average of the target values in the region). This is a continuous output.

- **Classification:** y^{R_j} = the most frequent class label among $\left\{y^{(j_1)}, \ldots, y^{(j_k)}\right\}$. This is a discrete output.

9.2 Regression decision tree

Consider "Hitters" dataset, where the goal is to predict baseball players' salaries based on their years of experience ("Years") and the number of hits they made in the previous year ("Hits").

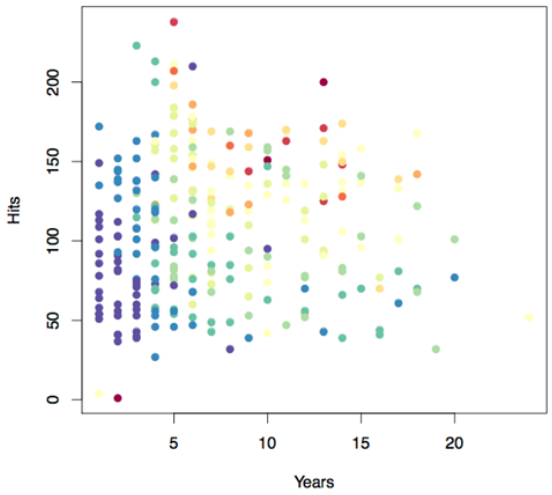

Figure 24: Distribution of the salaries

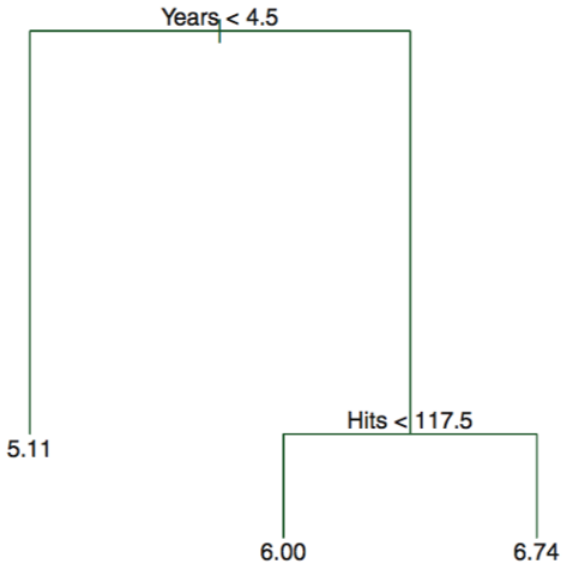

Figure 25: Example fitted tree

Each internal node shows a decision rule (e.g., "Years < 4.5"), and the branches represent the possible outcomes ("Yes" or "No"). The leaf nodes provide the predicted salary, which is the average salary of the players in that region.

And based on the decision tree above, the decision split the features into 3 regions:

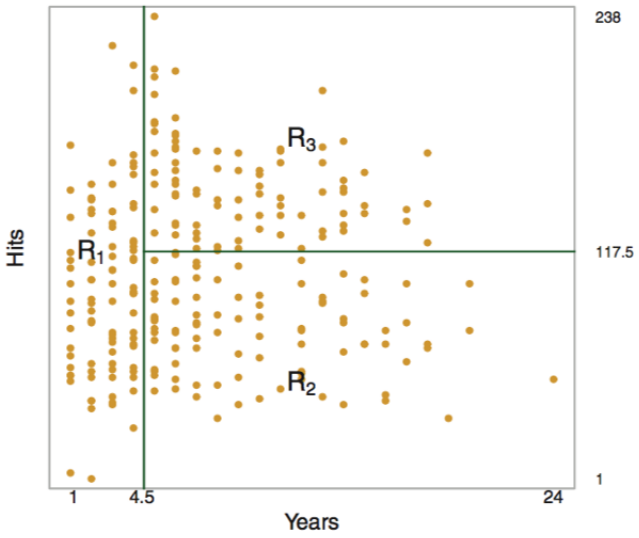

Figure 26: Separated feature space

- $R_1 = \{X \mid \text{Years} < 4.5\}$
- $R_2 = \{X \mid \text{Years} \geq 4.5, \text{ Hits} < 117.5\}$
- $R_3 = \{X \mid \text{Years} \geq 4.5, \text{ Hits} \geq 117.5\}$

From the separation, we can see that:

- Years is the most important factor in determining Salary, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of Hits that he made in the previous year seems to play little role in his Salary.
- But among players who have been in the major leagues for five or more years, the number of Hits made in the previous year does affect Salary, and players who made more Hits last year tend to have higher salaries.

Remark.

Compared to a regression model, DT is easier to interpret, and has a nice graphical representation.

9.2.1 Building a regression decision tree

Below are general steps to build a regression decision tree:

1. The feature space ($X_1, X_2, \ldots, X_p$) is divided into J non-overlapping regions ($R_1, R_2, \ldots, R_J$).
2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

Remark.

In theory, the regions $(R_1, R_2, \ldots, R_J)$ can take any shape, but for simplicity, they are constrained to be rectangular. Rectangular regions make the model easier to interpret.

9.2.2 Building regression DT: building regions

The goal of constructing the regions is minimizing the residual sum of squares (RSS) across all regions. The formula for RSS is:

$$\text{RSS} = \sum_{j=1}^{J} \sum_{i \in R_j} \left(y_i - \bar{y}_{R_j} \right)^2$$

where

$$\bar{y}_{R_j} = \frac{1}{|R_j|} \sum_{i \in R_j} y_i$$

is the mean response for the training observations in R_j.

Enumerating all possible ways to divide the feature space into J regions is computationally infeasible. Instead, a top-down, greedy approach is used, meaning:

- **Top-Down:** Start with the entire feature space.
- **Greedy:** At each step, choose the split that minimizes RSS without considering future splits.

The algorithm is computationally efficient and scalable.

At each step, the algorithm selects:

1. The best feature X_j among all features $X_1, X_2, \ldots, X_p$.
2. The best cutpoint s for splitting the feature in terms of RSS.

Mathematically, we seek the value of $j \in \{1, 2, \ldots, p\}$ and $s \in \mathbb{R}$ that minimize the equation:

$$\sum_{i: x_i \in R_1(j,s)} (y_i - \bar{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j,s)} (y_i - \bar{y}_{R_2})^2$$

- $R_1(j, s) = \{X \mid X_j < s\}$ and $R_2(j, s) = \{X \mid X_j \geq s\}$
- $\bar{y}_{R_1}$ and $\bar{y}_{R_2}$ are, respectively, the averaged responses of the training data in $R_1(j, s)$ and $R_2(j, s)$

The process of finding the best split is repeated iteratively:

1. Further split one of the previously identified regions.
2. Continue until a stopping criterion is met.

A common criterion is to stop when:

- No region contains more than five observations.
- Additional splits do not significantly reduce RSS.

Once the regions $R_1, R_2, \ldots, R_J$ are finalized, the prediction for a new data point is the average response value in the corresponding region.

Example.

Suppose we have X_1 and X_2. We choose the feature and threshold that minimizes the Residual Sum of Squares (RSS).

1. **First split:** If the split is based on $X_1 < t_1$, the feature space is divided into $R_1 = \{X \mid X_1 < t_1\}$ and $R_2 = \{X \mid X_1 \geq t_1\}$.
2. **Recursive split:** For each of the newly created regions (R_1, R_2), we choose the best feature and threshold within the region to further split the data.

   For example, in R_1, the next split might be based on $X_2 < t_2$, further dividing R_1 into $R_3 = \{X \mid X_1 < t_1, X_2 < t_2\}$ and $R_4 = \{X \mid X_1 < t_1, X_2 \geq t_2\}$.

   We keep splitting regions recursively until a stopping criterion is satisfied.

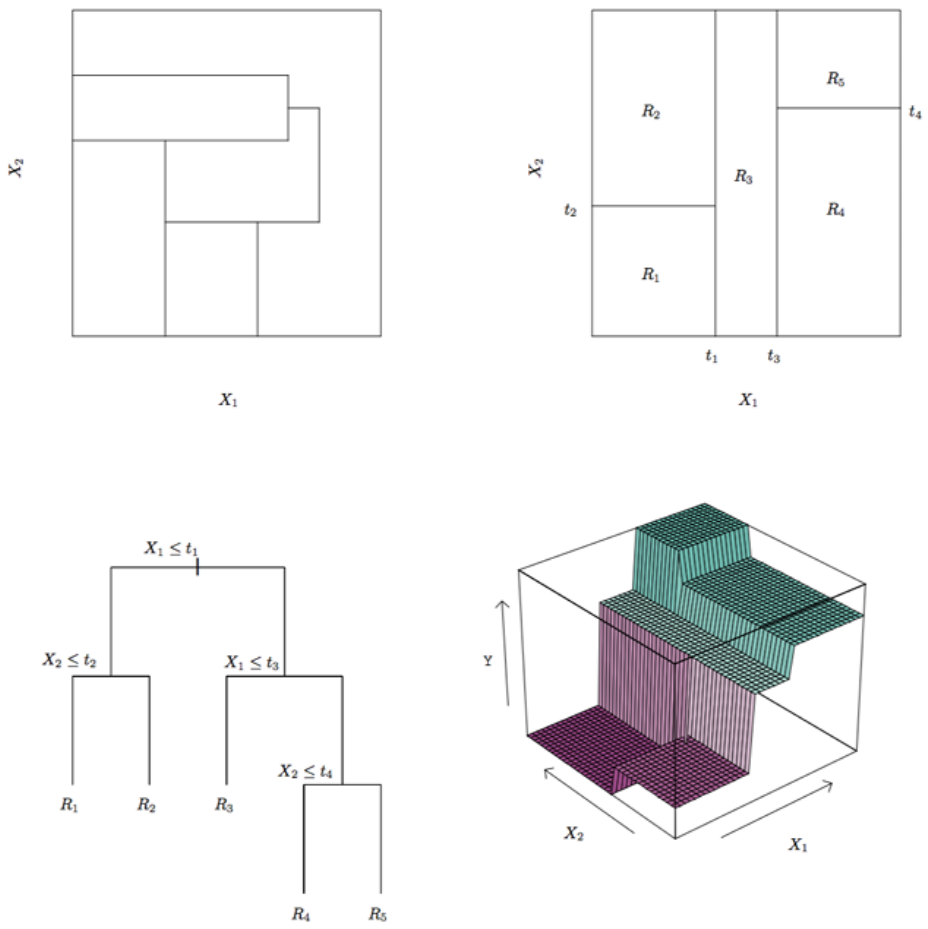

Figure 27: A example of 5 splits

- For the rectangle on top left, non-rectangular partitioning is not possible with the recursive binary splitting approach.
- For the rectangle on the top right, it shows the actual partitioning of feature space achieved using recursive binary splitting.
- And we have the decision tree corresponding to the recursive binary splits is shown.
- For the picture on bottom right, we see the prediction surface. Each region has a constant prediction value, which corresponds to the mean of the observations within that region.

Decision trees rely on recursive, axis-aligned splits, which creates rectangular decision boundaries. These boundaries simplify interpretation but might limit the ability to model complex relationships.

9.2.3 Tree pruning

A fully grown decision tree may fit the training data very well but generalize poorly to new data. For example, a tree that has one observation per region (overfitting) will have low bias but high variance, leading to poor performance on test data.

And the solution to this overffitting is tree pruning. Pruning involves simplifying the tree by removing unnecessary splits (i.e., reducing the number of regions $R_1, R_2, \ldots, R_J$). A smaller tree has:

- **Lower variance:** Reduces sensitivity to noise in the training data.
- **Higher bias:** May slightly oversimplify relationships but often generalizes better.

And the goal is to select a subtree that minimizes the test error rate. However, estimating test error for every possible subtree is computationally expensive due to the large number of possible subtrees.

A popular pruning approach is cost complexity pruning, also known as weakest link pruning. The process involves:

1. Growing a large tree (T_0).
2. Pruning back to smaller subtrees based on a cost complexity criterion.

For a given subtree T, the cost complexity is defined as:

$$\text{Cost Complexity } = \sum_{m=1}^{|T|} \sum_{i \in R_m} (y_i - \bar{y}_{R_m})^2 + \alpha |T|$$

- $|T|$: Number of terminal nodes (regions) in the subtree.
- α: Tuning parameter that controls the trade-off between model complexity and fit to the training data.
- $\bar{y}_{R_m}$: Mean response for region R_m.

Remark.

When $\alpha = 0$, there is no penalty for tree size; the algorithm retains the full tree (T_0).
As we have increasing α, we apply larger penalty for complexity, leading to smaller, pruned trees.

Remark.

As α increases from 0, branches are pruned from the tree in a predictable, nested sequence. This allows for the easy generation of a sequence of subtrees corresponding to different values of α.

Remark.

In order to choose the optimal α, we use a validation set or cross-validation to choose α that minimizes the test error rate. After selecting α, the corresponding subtree is obtained from the full tree (T_0).

9.2.4 Algorithm to built regression DT

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:
   (a) Repeat Steps 1 and 2 on all but the k th fold of the training data.
   (b) Evaluate the mean squared prediction error on the data in the left-out k th fold, as a function of α.

   Average the results for each value of α, and pick α to minimize the average error.
4. Return the subtree from step 2 that corresponds to the chosen value of α.

9.2.5 Complete example

Suppose we have a data set. We first randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set. We then built a large regression tree on the training data and varied α to create subtrees with different numbers of terminal nodes. Next, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of α.

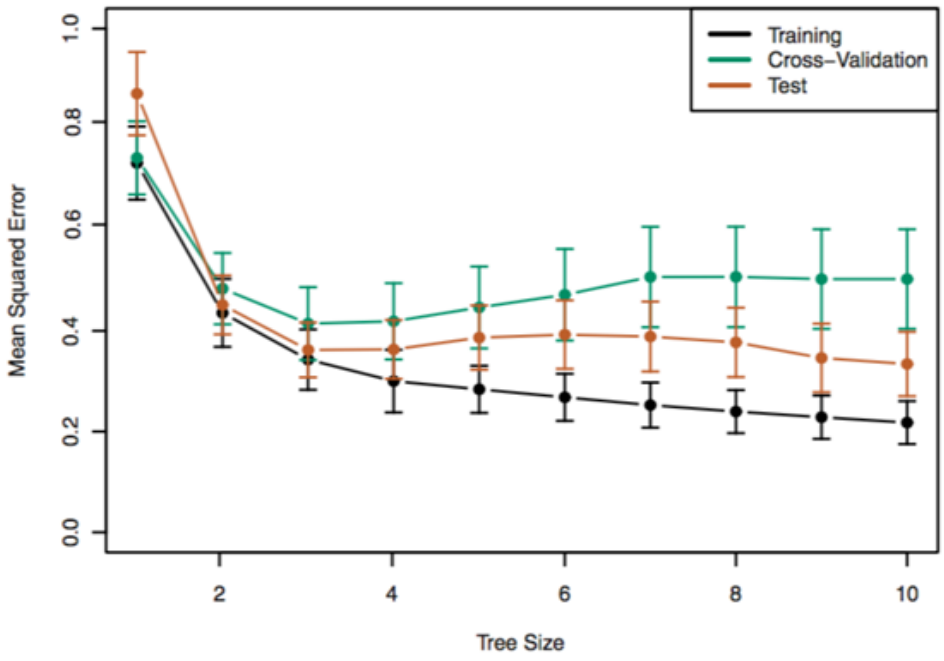

Figure 28: Cross-validation results

- **Training error (black line):** Decreases as tree size increases.
- **Cross-Validation error (orange line):** Initially decreases but begins to increase as the tree becomes too large (overfitting).
- **Test error (green line):** Mimics cross-validation error, reaching its minimum at the optimal tree size.

The optimal tree size is chosen where cross-validation error is minimized. And overfitting is evident when tree size is too large, as it leads to increased test error.

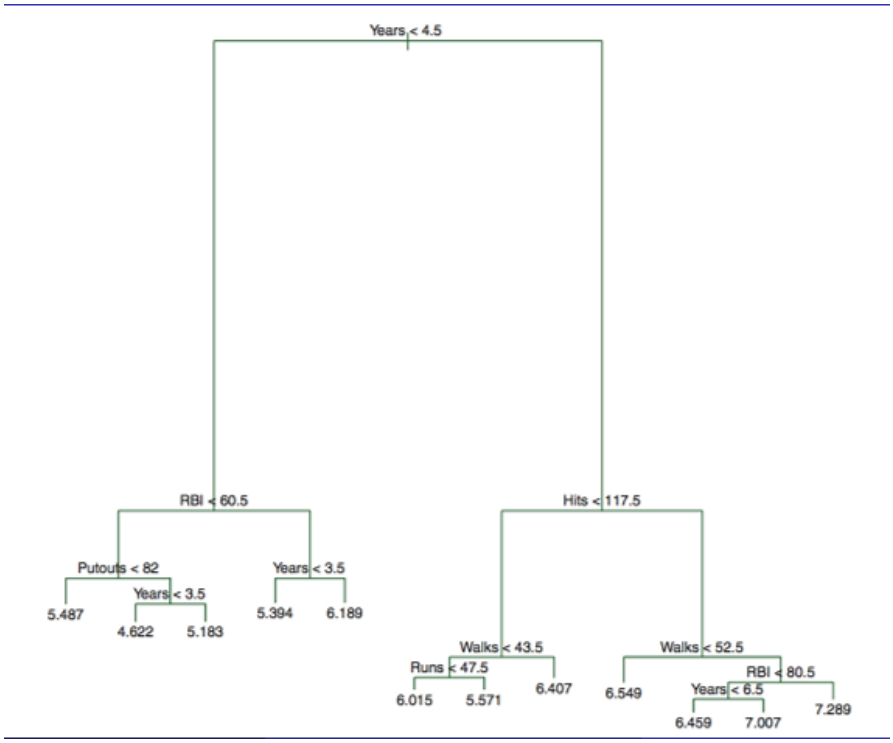

Figure 29: Unpruned tree

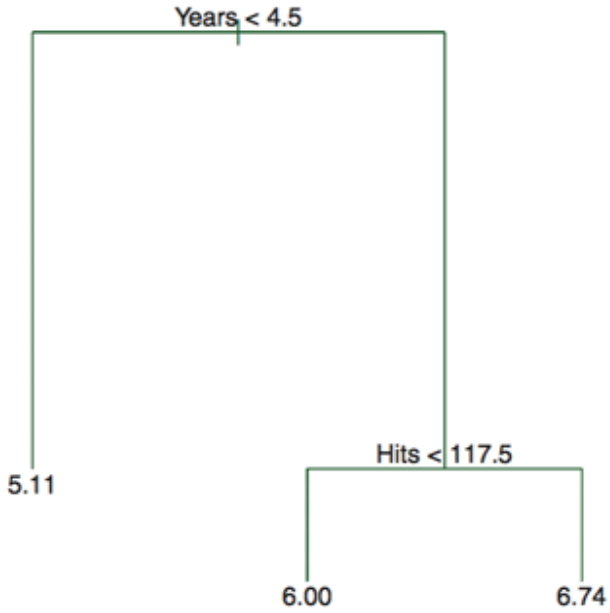

Figure 30: Pruned tree

9.3 Classification tree

The prupose of classification tree is to predict qualitative (categorical) responses instead of continuous values. Each region (terminal node) predicts the majority class among the observations in that region. Like regression trees, classification trees use recursive binary splitting to partition the feature space.

In regression trees, the splitting criterion is based on minimizing residual sum of squares (RSS). For classification trees, RSS is not applicable. Thus, a new metric is required to evaluate the quality of splits.

9.3.1 Final metrics used in classification tree

Definition 9.6: $0-1$ misclassification error rate

The $0-1$ misclassification error rate measures the proportion of incorrect predictions in a region. For region R_m with K classes, the misclassification error is:

$$1 - \max_{k \in \{1,\ldots,K\}} \hat{p}_{mk}$$

where $\hat{p}_{mk} = \frac{1}{|R_m|} \sum_{i:x_i \in R_m} 1\{y_i = k\}$.
And Overall misclassification error $= \sum_{m=1}^{M} \frac{|R_m|}{N} \cdot$ misclassification error in regionR_m

However, misclassification error is not sensitive enough to small changes in class proportions. It is better suited for evaluating the final tree rather than selecting splits during tree construction. Thus, we need more metrics.

9.3.2 Selection metrics used in classification tree

Example.

Consider the following example:

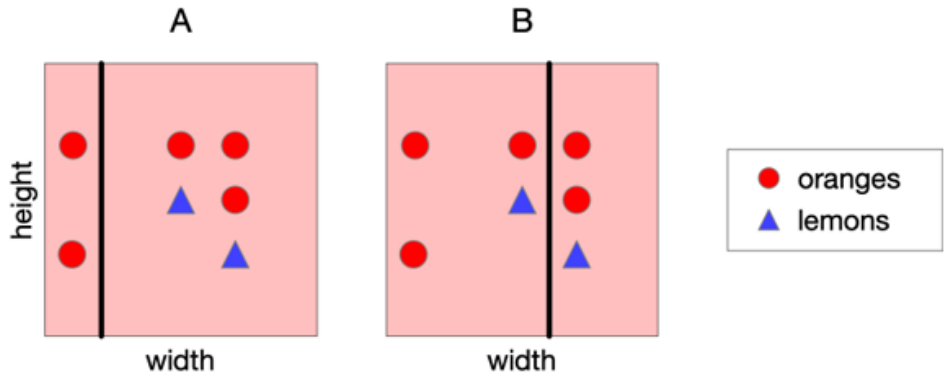

Figure 31: Two different splits

1. For A:
   - $P_{L,\text{orange}} = \frac{2}{2} = 1$
   - $P_{L,\text{lemons}} = \frac{0}{2} = 0$
   - Left error $= 1 - \max(0, 1) = 1 - 1 = 0$
   - $P_{R,\text{orange}} = \frac{3}{5}$
   - $P_{R,\text{lemons}} = \frac{2}{5}$
   - Right error $= 1 - \max(\frac{3}{5}, \frac{2}{5}) = \frac{2}{5}$
   - Overall error $= \frac{2}{7} \times 0 + \frac{5}{7} \times \frac{2}{5} = \frac{2}{7}$
2. For B:
   - $P_{L,\text{orange}} = \frac{3}{4}$
   - $P_{L,\text{lemons}} = \frac{1}{4}$
   - Left error $= 1 - \max(0, 1) = 1 - \frac{3}{4} = \frac{1}{4}$
   - $P_{R,\text{orange}} = \frac{2}{3}$
   - $P_{R,\text{lemons}} = \frac{1}{3}$
   - Right error $= 1 - \max(\frac{2}{3}, \frac{1}{3}) = \frac{1}{3}$
   - Overall error $= \frac{4}{7} \times \frac{1}{4} + \frac{3}{7} \times \frac{1}{3} = \frac{2}{7}$

Both have the same misclassification rate, but one may better reflect the certainty of classification. Split A feels better because it creates regions with more purity (i.e., regions dominated by a single class). And the key question is can we quantify the advantage of Split A over Split B?

9.3.3 New metrics: entropy

Definition 9.7: Entropy

Entropy is a metric from information theory that quantifies the uncertainty in a discrete random variable's outcomes.

- High entropy indicates high uncertainty (e.g., a balanced mix of classes).
- Low entropy indicates low uncertainty or high purity (e.g., a region dominated by one class).

The entropy H for a binary classification problem (two classes) is:

$$H = -p \log_2(p) - (1-p) \log_2(1-p)$$

where p is the proportion of one class in a region.

Remark.

The use of entropy is more sensitive to changes in class proportions than misclassification error, making it suitable for finding the best splits during the selection process.

Example.

Back to our previous example:

1. For split A:
   - Left Branch: $p_{\text{orange}} = 1, p_{\text{lemon}} = 0$

     $$H = -1 \log_2 1 - 0 \log_2 0 = 0$$

   2. Right Branch: $p_{\text{orange}} = \frac{3}{5}, p_{\text{lemon}} = \frac{2}{5}$

      $$H = -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} \approx 0.97$$

   3. Total Entropy for Split A: $H = 0 + 0.97 = 0.97$.
2. For split B:
   - Left branch: $p_{\text{orange}} = \frac{3}{4}, p_{\text{lemon}} = \frac{1}{4}$

     $$H = -\frac{3}{4} \log_2 \frac{3}{4} - \frac{1}{4} \log_2 \frac{1}{4} \approx 0.81$$

   - Right branch: $p_{\text{orange}} = \frac{2}{3}, p_{\text{lemon}} = \frac{1}{3}$

     $$H = -\frac{2}{3} \log_2 \frac{2}{3} - \frac{1}{3} \log_2 \frac{1}{3} \approx 0.92$$

   - Total entropy for split B: $H = 0.81 + 0.92 = 1.73$.

Split A (Total Entropy $= 0.97$) achieves a lower entropy than Split B (Total Entropy $= 1.73$), indicating greater node purity and better classification certainty.

9.3.4 New metrics: Deviance and Gini index

Definition 9.8: Deviance

Deviance is the entropy-based metric used in classification problems.

$$\text{Deviance } = -\sum_{k=1}^{K} \hat{p}_{mk} \log\left(\hat{p}_{mk}\right)$$

Deviance is small when the node is pure (most observations belong to a single class).

Definition 9.9: Gini index

The Gini index is a measure of total variance across classes in a region.

$$\text{Gini index } = \sum_{k=1}^{K} \hat{p}_{mk}\left(1 - \hat{p}_{mk}\right)$$

The Gini index is small when most observations in a node belong to a single class.

Remark.

Both Deviance and Gini index are measures of node purity. Smaller values of these metrics indicate purer nodes.

Remark.

For building a classification tree, either Gini index or Deviance can be used for finding best splits, since they are more sensitive to node purity than the misclassification error rate.
On the other hand, all Gini index, Deviance and misclassification error can be used for pruning the tree, but the misclassification error is preferable if classification accuracy is the final goal.

9.4 Tree method vs. linear model

A linear model predicts the target variable Y as a weighted sum of features:

$$f(X) = \beta_0 + \sum_{j=1}^{p} \beta_j X_j$$

A regression tree makes predictions by dividing the feature space into J non-overlapping regions $R_1, R_2, \ldots, R_J$, with each region having a constant prediction:

$$f(X) = \sum_{m=1}^{J} c_m \cdot 1\left\{X \in R_m\right\}$$

- c_m: Mean response value in region R_m.
- $1\left\{X \in R_m\right\}$: Indicator function that equals 1 if X belongs to region R_m, and 0 otherwise.

If $f(X) \approx X^\top \boldsymbol{\beta}$, then linear regression will likely work well. If instead there is a highly non-linear and complex relationship between X and Y, then decision trees performs better.

The relative performances of tree-based and classical approaches can be assessed by estimating the test error, using either cross-validation or the validation set approach.

9.5 Pros and Cons for DT

1. Advantages:

- **Interpretability:** Decision trees are highly interpretable and visually intuitive. They are easier to explain than linear models, especially for non-technical audiences.
- **Human-Like Decisions:** Decision trees mimic human decision-making processes, making their logic relatable.
- **Handling Qualitative Features:** Trees can handle categorical (qualitative) features without requiring one-hot encoding or dummy variables.

2. Disadvantages:

- **Predictive Power:** Decision trees generally have lower predictive accuracy compared to other methods like ensemble models.
- **Non-Robustness:** Trees are sensitive to small changes in the data. A minor variation in the dataset can lead to significantly different tree structures.
- **Overfitting:** Fully grown trees often overfit the training data, leading to poor generalization.

10 Bagging, random forest, and boosting

We will address the limitations of individual machine learning models, especially decision trees, by introducing ensemble methods that combine multiple models to improve performance.

We know for a decision tree, if grown without constraints, decision trees can capture a lot of noise in the data, leading to overfitting. And small changes in the data can lead to drastically different tree structures and predictions. To mitigate overfitting, trees are pruned after being grown.

10.1 Bootstrap and its applications

Definition 10.1: Bootstrapping

Bootstrap is a resampling technique used to estimate the variability and distribution of a statistic (e.g., mean, variance, coefficients) from a dataset. The purpose is to quantify the uncertainty of a statistical procedure.

Example.

Bootstrap can estimate standard errors and the whole distribution of estimated coefficients in linear regression.

10.1.1 A simple example

Example.

Considering the following example. You wish to invest $\$10,000$ in two assets X and Y. Fraction $\alpha \in [0,1]$, is invested in X, and the rest $(1-\alpha)$ is invested in Y. The total return would be $[\alpha X + (1-\alpha)Y] \times 10,000$. We wish to choose α to minimize the total risk, or variance, of our investment:

$$\min_{\alpha \in [0,1]} \operatorname{Var}(\alpha X + (1-\alpha)Y)$$

The portfolio return is given as:

$$R = \alpha X + (1-\alpha)Y$$

The variance of R is:

$$\operatorname{Var}(R) = \operatorname{Var}(\alpha X + (1-\alpha)Y)$$

We expand this expression:

$$\operatorname{Var}(R) = \alpha^2 \operatorname{Var}(X) + (1-\alpha)^2 \operatorname{Var}(Y) + 2\alpha(1-\alpha)\operatorname{Cov}(X,Y)$$

Substituting $\operatorname{Var}(X) = \sigma_X^2$, $\operatorname{Var}(Y) = \sigma_Y^2$, and $\operatorname{Cov}(X,Y) = \sigma_{XY}$, we get:

$$\operatorname{Var}(R) = \alpha^2\sigma_X^2 + (1-\alpha)^2\sigma_Y^2 + 2\alpha(1-\alpha)\sigma_{XY}$$

Expand the terms:

$$(1-\alpha)^2 = 1 - 2\alpha + \alpha^2,$$

so:

$$\operatorname{Var}(R) = \alpha^2\sigma_X^2 + \left(1 - 2\alpha + \alpha^2\right)\sigma_Y^2 + 2\alpha(1-\alpha)\sigma_{XY}$$

$$\operatorname{Var}(R) = \alpha^2\sigma_X^2 + \sigma_Y^2 - 2\alpha\sigma_Y^2 + \alpha^2\sigma_Y^2 + 2\alpha\sigma_{XY} - 2\alpha^2\sigma_{XY}$$

$$\operatorname{Var}(R) = \alpha^2\left(\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}\right) - 2\alpha\left(\sigma_Y^2 - \sigma_{XY}\right) + \sigma_Y^2$$

To minimize $\operatorname{Var}(R)$, we take the derivative with respect to α and set it to zero:

$$\frac{d}{d\alpha}\operatorname{Var}(R) = 2\alpha\left(\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}\right) - 2\left(\sigma_Y^2 - \sigma_{XY}\right)$$

Set the derivative to zero:

$$2\alpha\left(\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}\right) - 2\left(\sigma_Y^2 - \sigma_{XY}\right) = 0$$

Simplify:

$$\alpha\left(\sigma_X^2+\sigma_Y^2-2\sigma_{XY}\right)=\sigma_Y^2-\sigma_{XY}$$

Solve for α:

$$\alpha=\frac{\sigma_Y^2-\sigma_{XY}}{\sigma_X^2+\sigma_Y^2-2\sigma_{XY}}$$

To confirm that this is a minimum, we check the second derivative:

$$\frac{d^2}{d\alpha^2}\operatorname{Var}(R)=2\left(\sigma_X^2+\sigma_Y^2-2\sigma_{XY}\right)$$

Since $\sigma_X^2+\sigma_Y^2-2\sigma_{XY}>0$, the second derivative is positive. Thus, the critical point is a minimum.
As we have shown, the variance-minimizing value of α is derived as:

$$\alpha=\frac{\sigma_Y^2-\sigma_{XY}}{\sigma_X^2+\sigma_Y^2-2\sigma_{XY}},$$

However, in real scenarios, $\sigma_{X'}^2,\sigma_{Y'}^2$ and σ_{XY} must be estimated from historical data. When historical data is available, the optimal α is estimated as:

$$\hat{\alpha}=\frac{\hat{\sigma}_Y^2-\hat{\sigma}_{XY}}{\hat{\sigma}_X^2+\hat{\sigma}_Y^2-2\hat{\sigma}_{XY}}$$

where $\hat{\sigma}_X^2,\hat{\sigma}_Y^2,\hat{\sigma}_{XY}$ are sample estimates of variance and covariance.
The key question remains on how do we estimate the variance of $\hat{\alpha}$? And oen potential approach is the Oracle approach.

Example.

Assuming a unrealistic assumption that the distributions of X and Y are known. We would:

1. We simulate 100 paired observations of X and Y and compute

$$\hat{\alpha}=\frac{\hat{\sigma}_Y^2-\hat{\sigma}_{XY}}{\hat{\sigma}_X^2+\hat{\sigma}_Y^2-2\hat{\sigma}_{XY}}$$

2. We repeat this procedure 1000 times, and get $\hat{\alpha}_1,\ldots,\hat{\alpha}_{1000}$.
3. Estimate $\operatorname{Var}(\hat{\alpha})$ using the sample variance:

$$\hat{\operatorname{Var}}(\hat{\alpha})=\frac{1}{999}\sum_{r=1}^{1000}\left(\hat{\alpha}_r-\bar{\alpha}\right)^2,$$

   where $\bar{\alpha}$ is the mean of the simulated $\hat{\alpha}_r$.

In practice, the true distributions of X and Y are rarely known, making this approach infeasible.

Example.

Bootstrap provides a practical alternative to the oracle approach by resampling directly from the observed data. Here are the procedures:

1. Generate B bootstrap samples (e.g., $B=1000$) from the original dataset by sampling with replacement.
2. Compute $\hat{\alpha}_b^*$ for each bootstrap sample b.
3. Estimate $\operatorname{Var}(\hat{\alpha})$ as:

$$\hat{\operatorname{Var}}(\hat{\alpha})=\frac{1}{B-1}\sum_{b=1}^{B}\left(\hat{\alpha}_b^*-\bar{\alpha}^*\right)^2,$$

where $\bar{\alpha}^* = \frac{1}{B}\sum_{b=1}^{B}\hat{\alpha}_b^*$.

Note that as a result, data set from bootstrap might contain some observations more than once, or zero time.

Remark.

Bootstrap mimics the process of obtaining new datasets and provides a feasible way to estimate parameter variability. And it is a reliable alternatives.

10.1.2 Bootstrap for OLS estimation

Bootstrap can also be applied to assess the uncertainty of Ordinary Least Squares (OLS) regression coefficients. Given data $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, the OLS estimator is:

$$\hat{\beta} = \left(X^\top X\right)^{-1} X^\top y$$

Statistical properties of $\hat{\beta}$ include:

- Mean
- Covariance
- Higher moments or entire distribution

All of them can be estimated using bootstrap.

Here, we will follow the following procedures:

1. Generate B bootstrap samples $(D_1, D_2, \ldots, D_B)$.
2. For each bootstrap sample b, we compute $\hat{\beta}_b^* = \left(X_b^\top X_b\right)^{-1} X_b^\top y_b$.
3. Use the bootstrap estimates $\left\{\hat{\beta}_1^*, \ldots, \hat{\beta}_B^*\right\}$ to:
   - Estimate the mean of β_j as:
   $$\hat{\mu}_{\beta_j} = \frac{1}{B}\sum_{b=1}^{B}\hat{\beta}_{b,j}^*$$
   - Estimate the variance of β_j as:
   $$\hat{\sigma}_{\beta_j}^2 = \frac{1}{B-1}\sum_{b=1}^{B}\left(\hat{\beta}_{b,j}^* - \hat{\mu}_{\beta_j}\right)^2$$
   - Estimate quartiles of distribution of $\hat{\beta}_j$ as:
     - Sort the bootstrap estimates $\left\{\hat{\beta}_{1,j}^*, \hat{\beta}_{2,j}^*, \ldots, \hat{\beta}_{B,j}^*\right\}$ in ascending order.
     - To find the q-th quantile (e.g., 2.5 th or 97.5 th percentile), we identify the index corresponding to $q \times B$, where B is the number of bootstrap samples.
     For example, if $B = 1000$ and $q = 0.025$, the 2.5 th quantile is at the 25 th value in the sorted bootstrap sample.
   - Estimate the sampling distribution is simply the collection $\left\{\hat{\beta}_{1,j}^*, \hat{\beta}_{2,j}^*, \ldots, \hat{\beta}_{B,j}^*\right\}$, and it is treated as an empirical approximation of the sampling distribution of $\hat{\beta}_j$.

10.2 Bagging

Definition 10.2: Bagging

Bagging, short for "Bootstrap Aggregating," is an ensemble method designed to improve the performance of machine learning models, particularly those prone to high variance (e.g., decision trees).
Suppose we have N independent estimates of a parameter β:

$$\hat{\beta}^{(1)}, \hat{\beta}^{(2)}, \ldots, \hat{\beta}^{(N)}$$

If each individual estimate has bias $\left|E\left[\hat{\beta}^{(i)}\right] - \beta\right| \leq b$ and variance $\mathrm{Var}\left(\hat{\beta}^{(i)}\right) = \sigma^2$, then the averaged estimate:

$$\bar{\beta} = \frac{1}{N}\sum_{i=1}^{N} \hat{\beta}^{(i)}$$

has bias $|E[\bar{\beta}] - \beta| \leq b$, and variance $\mathrm{Var}(\bar{\beta}) = \frac{\sigma^2}{N}$.
Thus, averaging multiple estimates reduces variance without increasing bias.

10.2.1 Bagging in DT

Bagging is particularly useful for decision trees, which are highly sensitive to small changes in the training data. We following the procedures below:

1. Generate B bootstrap samples $(D_1, D_2, \ldots, D_B)$ from the training dataset.
2. Train a decision tree on each bootstrap sample D_b. Denote the prediction of the tree trained on D_b as $\hat{f}_b^*(x)$.
3. For regression:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b^*(x).$$

   For classification: Take a majority vote of the predictions from all B trees.

Decision trees in bagging are typically grown deep and unpruned, resulting in high variance but low bias. By averaging predictions, bagging reduces the variance of the overall model. For a test observation, record the predicted class from each of the B trees and use a majority vote for the final prediction.

10.2.2 Out-of-Bag (OOB) error estimation

Bagging provides a built-in way to estimate the test error without requiring a separate validation set or cross-validation: the Out-of-Bag (OOB) error estimation.

Each bootstrap sample D_b contains approximately $2/3$ of the original observations. The remaining $1/3$ of the observations are referred to as the **OOB observations**. The OOB observations for a given tree are not used in its training.

1. For each observation i: Collect predictions from all trees where i was part of the OOB set.
2. Aggregate these predictions:
   - **For regression:** Compute the average of the predictions.
   - **For classification:** Take a majority vote of the predictions.
3. Compute the Mean Squared Error (MSE) or misclassification error using the aggregated predictions.

Thus, we eliminates the need for a separate validation set. And we provides an unbiased estimate of the test error.

10.2.3 Variable importance

Bagging improves prediction accuracy but can make the model harder to interpret. Variable importance measures help provide insights into which predictors are most influential.

- For Regression Trees: it records the total reduction in the RSS due to splits involving a given predictor, averaged over all B trees.
- For Classification Trees: it replace RSS with the Gini index (a measure of class purity). And record the total reduction in the Gini index due to splits involving each predictor.

A larger reduction in RSS (or Gini index) indicates that the predictor is more important for the model. Results can be visualized in a bar chart to rank predictor importance. For example:

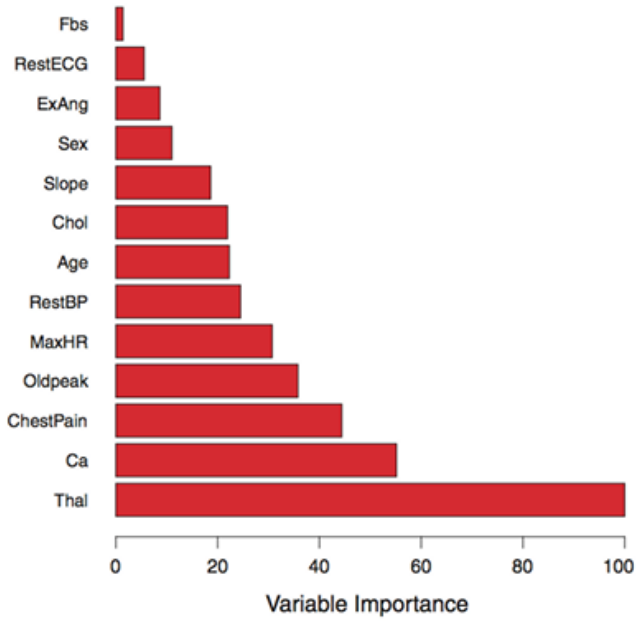

Figure 32: Variable importance example

- The top predictors (e.g., "Thal," "Ca," "ChestPain") contribute the most to the model's performance.
- Less important predictors (e.g., "Fbs," "RestECG") have smaller impacts.

10.3 Random forest

The key limitation of bagging is if some predictors are strong (highly predictive), all bagged trees might use these predictors in their top splits, making the trees highly correlated. And on the other hand, random forests extend Bagging by introducing additional randomness during tree construction, which reduces the correlation among trees and further improves the ensemble's performance.

Here is the procedures of random forest:

1. As in bagging, we wish to build a number of decision trees on bootstrap training samples.
2. At each split in a tree, only a random subset of m predictors is considered (instead of all p predictors) when training.

   The best split is chosen from this subset.

   Typical choice: $m = \sqrt{p}$, where p is the total number of predictors.
3. Prediction Aggregation:
   - **For regression:** Average the predictions from all trees.
   - **For classification:** Use a majority vote.

In a dataset with one strong predictor and several moderately strong predictors, Bagging may overuse the strong predictor in top splits of all trees, leading to correlated predictions.

Random Forests force the trees to consider different subsets of predictors at each split, reducing correlation. As a result, random forests produce more diverse trees, which reduces the ensemble's variance without increasing bias.

Remark.

Random Forest with $m = p$ is equivalent to Bagging.

Example.

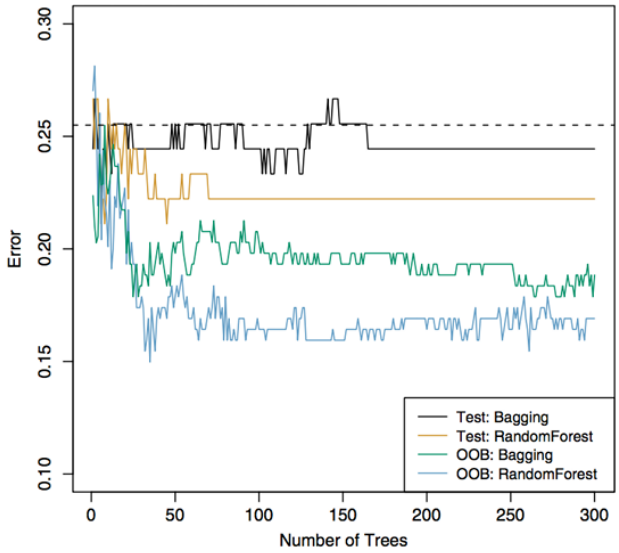

Figure 33: Example with heart data

Random Forests offer better predictive performance and lower error rates compared to Bagging, particularly when the number of trees is large.

Remark.

Random Forests are particularly effective for high-dimensional data, where they improve accuracy by reducing overfitting.

10.4 Boosting

Boosting is another ensemble technique that improves model performance by sequentially correcting errors made by earlier models. The key idea is that unlike Bagging or Random Forests, where trees are trained independently, Boosting builds trees sequentially. Each tree is trained to focus on the residual errors of the previous trees.

However, unlike bagging, no bootstrap sampling is used; each tree is trained on a modified version of the original dataset. Trees are grown sequentially, with each new tree improving on the performance of the previous ensemble.

The algorithm for Boosting is given by:

1. **Initialization:** Start with an initial prediction $\hat{f}(x) = 0$. Set residuals $r_i = y_i$ for all i (the actual response values).

2. **Iterative training:** For $b = 1, \ldots, B$:

    - Fit a tree $f^b(x)$ to the residuals r_i.

- Update the prediction: Add a shrunken version of the new tree to the model:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda f^b(x),$$

  where λ (the shrinkage parameter) controls the learning rate.
- Update residuals: Adjust residuals to reflect the new predictions:

$$r_i \leftarrow r_i - \lambda f^b\left(x_i\right)$$

3. After B iterations, the final boosted model is:

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda f^b(x)$$

Boosting has three key hyperparameters:

- Number of Trees (B): it controls the complexity of the ensemble. However, boosting can overfit if B is too large. We use cross-validation to select B.
- Shrinkage Parameter (λ): it Controls the contribution of each tree. The typical values: 0.01 or 0.001. And smaller λ slows learning and reduces the risk of overfitting.
- Tree Depth (d): it limits the number of splits in each tree. Commonly, $d = 1$ (stumps) works well, producing an additive model.