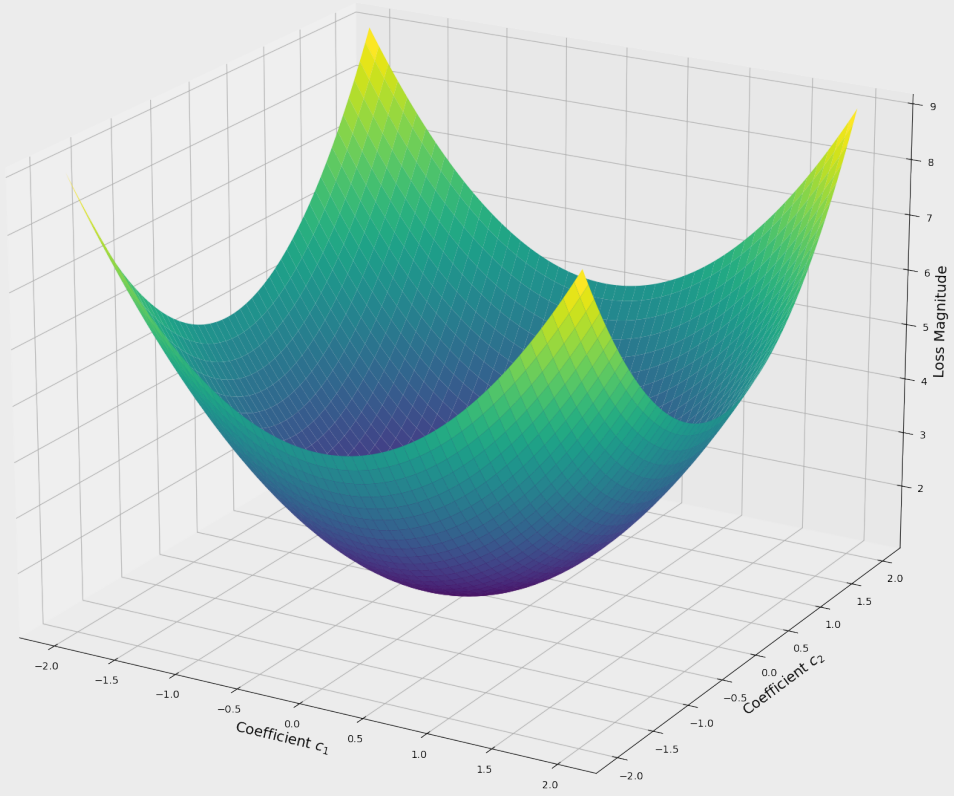
Mastering Data Science Interviews

Explain Any Concept in Data Science, Machine Learning and Artificial Intelligence



William Alston

Copyright

First published in 2026

coherentnoise publishing
coherentnoise.space

Contents

1 The Aim of This Book **1**

1.1 Why This Book Exists . . . 1
1.2 The Challenge of Technical Interviews . . . 2
1.3 The Gap Between Coursework and Interviews . . . 2
1.4 Learning Through Interview Questions . . . 3
1.5 What This Book Assumes . . . 3
1.6 The Five Types of Understanding . . . 4
1.7 How Each Question Is Structured . . . 5
1.8 How to Use This Book . . . 7
1.9 Question Difficulty Levels . . . 7
1.10 A Final Note . . . 8

I Modelling and Theory **9**

2 The Bias–Variance trade-off **11**

II Optimisation and Training **29**

3 Loss Functions **31**

4 Gradient Descent **69**

III Evaluation and Metrics **90**

5 Information Criteria **92**

6 Statistical Testing **104**

IV Conceptual Foundations **118**

7 Maximum Likelihood Estimation **120**

8 Generative vs Discriminative Models **126**

V Applied ML and Systems **133**

9 Production ML Systems **135**

Preface

This book is intended to help Master's students prepare seriously and confidently for technical interviews in data science, machine learning, and artificial intelligence. It goes beyond short revision-style answers by developing the mathematical ideas, intuitive understanding, and practical interpretation that sit behind common interview topics. The goal is to help readers respond in a way that is not only correct, but also clear, well-structured, and convincing in an interview setting.

My own perspective comes from working across research and teaching for many years. I am an astronomer and a lecturer in data science, and over the past twenty years I have used machine learning in both scientific practice and higher education. Through that experience, I have seen how often students know the vocabulary of machine learning without yet feeling able to explain it fluently or apply it with confidence. This book is my attempt to bridge that gap by bringing together theory, intuition, and implementation in a form designed specifically for technically strong students preparing for demanding interviews.

Chapter 1

The Aim of This Book

1.1 Why This Book Exists

Over the past decade, the demand for data scientists, machine learning engineers, and artificial intelligence specialists has grown dramatically. Organisations across technology, finance, healthcare, retail, and government increasingly rely on data-driven decision making and predictive modelling. As a result, universities around the world now offer Masters degrees in areas such as:

- Data Science
- Machine Learning
- Artificial Intelligence
- Statistics
- Applied Mathematics
- Computer Science

Students graduating from these programmes typically have strong technical training. They study probability, statistical inference, machine learning algorithms, optimisation, and programming. They complete projects involving real datasets and build predictive models using modern tools such as Python and its scientific libraries.

However, when many graduates begin applying for jobs, they encounter a challenge that their coursework may not have fully prepared them for: the technical interview. Based on decades of experience in development and applications of machine learning techniques to scientific problems and designing several Masters level modules, I have developed this guide in order for newly graduated students to land that dream data scientist role. This guide will also be a valuable resource for any student about to give their Masters coursework viva.

1.2 The Challenge of Technical Interviews

Technical interviews for data science and machine learning roles rarely focus only on whether a candidate has seen a concept before. Instead, interviewers want to determine whether the candidate truly understands the ideas behind the methods they use. Candidates are often asked questions such as:

- What is the bias–variance trade-off?
- Why does logistic regression use the sigmoid function?
- What is the difference between L1 and L2 regularisation?
- How does gradient boosting work?
- What assumptions does linear regression make?
- What does the Central Limit Theorem tell us?

At first glance, these questions appear straightforward. Many students recognise the topics immediately. However, recognising a concept and *explaining it clearly and rigorously* are very different skills.

Many candidates discover during interviews that they can recall definitions but perhaps they struggle to explain the intuition behind an algorithm, the mathematical reasoning that motivates it, the assumptions required for it to work, or the the situations where it may fail. This book was written to help bridge that gap.

Interview Tip

Technical interviews are not only testing whether you know an algorithm. They are testing whether you understand *why it works*, *when it should be used*, and *how to explain it clearly.*

Throughout the book, these coloured text boxes will appear to offer additional advice and tips on intervews, mathematical insights and where to go next for a deeper dive on a topic.

1.3 The Gap Between Coursework and Interviews

University and college courses typically focus on teaching the theory and implementation of statistical and machine learning methods. Students learn how to apply algorithms to datasets and evaluate their performance. Once they have gained this understanding, students will typically encounter *real world* data problems in a non-production setting.

This training is essential. However, interviews require an additional skill: the ability to explain technical concepts clearly and logically under pressure. Interviewers often explore a candidate's understanding by asking follow-up questions such as:

- Why does this algorithm work?
- What assumptions does the model make?
- When would you choose this method instead of another?
- How would you diagnose a model that is performing poorly?

These questions reveal whether a candidate truly understands the principles behind machine learning models.

Deep Dive

A strong candidate is not someone who can simply name many algorithms. A strong candidate understands the principles that connect them: optimisation, probability, statistical inference, and linear algebra. Interviews often explore these underlying ideas.

1.4 Learning Through Interview Questions

The central idea of this book is simple:

The best way to prepare for data science interviews is to learn through the questions that interviewers actually ask.

Rather than presenting machine learning purely as an academic subject, this book approaches the material through common interview questions used by companies hiring data scientists and machine learning engineers. Each question becomes an opportunity to explore both the intuition and the mathematics behind the method.

1.5 What This Book Assumes

This book is written primarily for students who have completed, or are currently completing, a Master's degree in data science or a related discipline. Readers should already be familiar with:

- basic probability and statistics
- linear algebra
- Python programming
- fundamental machine learning concepts

The goal of the book is not to introduce these topics from the beginning, but to deepen understanding and prepare students to explain these ideas clearly in technical interviews.

1.6 The Five Types of Understanding

Machine learning interviews often appear to cover a wide range of unrelated topics. In a single interview, a candidate might be asked about linear regression, gradient descent, cross-validation, Bayesian inference, and system design. At first glance these questions may seem disconnected.

In practice, however, most machine learning interview questions fall into a small number of conceptual categories. Interviewers are usually trying to assess whether a candidate understands:

- ✓ how machine learning models represent relationships in data
- ✓ how these models are trained and optimised
- ✓ how their performance is evaluated
- ✓ the statistical ideas that underpin these methods
- ✓ how machine learning systems behave in real-world environments

This book is organised around these five types of knowledge.

Part I: Modelling and Theory introduces the core machine learning models that appear most frequently in interviews. These chapters focus on understanding how different modelling approaches represent patterns in data and how their behaviour relates to concepts such as bias, variance, and model complexity.

Part II: Optimisation and Training explains how machine learning models learn from data. Even a simple model can behave in surprising ways during training, and many interview questions focus on optimisation algorithms, regularisation, and training dynamics.

Part III: Evaluation and Metrics examines how models are assessed and compared. In practical machine learning, it is not enough to train a model; we must also determine whether it generalises well and whether its predictions can be trusted. This section explores evaluation techniques and common pitfalls such as data leakage.

Part IV: Conceptual Foundations develops the statistical ideas that underlie machine learning. Many algorithms can be understood more clearly when viewed through the lens of probability, estimation theory, and statistical reasoning.

Part V: Applied and Systems focuses on the challenges that arise when machine learning is used in real-world systems. Interviewers frequently ask questions about handling imperfect data, deploying models in production, and designing large-scale machine learning systems.

Together, these five parts reflect the different dimensions of knowledge that strong machine learning practitioners are expected to possess. The aim of this structure is not only to prepare

readers for interview questions, but also to help them build a coherent understanding of how machine learning models are developed, evaluated, and used in practice.

1.7 How Each Question Is Structured

Each interview question in this book follows a consistent structure designed to help readers develop both concise explanations and deeper technical understanding.

The Interview Question

Each section begins with a question that commonly appears in technical interviews.

Interview Question

What is the bias–variance trade-off?

Short Interview Answer

In an interview setting, candidates must first give a concise explanation, usually within one or two minutes. Each section therefore begins with a short answer that demonstrates how a strong candidate might respond.

Short Interview Answer

The bias–variance trade-off describes the balance between a model that is too simple to capture the underlying structure of the data and one that is so flexible that it overfits noise in the training data.

Intuition

A clear conceptual explanation helps interviewers see that the candidate understands the idea rather than simply memorising a definition. This often involves some mathematical reasoning.

Mathematical Foundations

Machine learning and data science are built on mathematical ideas from linear algebra, probability theory, statistics and optimisation. Each topic therefore includes a deeper explanation of the mathematical foundations underlying the concept, as can be seen in this example on polynomial regression.

Mathematical Insight

Model complexity is often related to the number of parameters in a model.
Consider a polynomial regression model of degree d:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_d x^d.$$

This model contains $d + 1$ parameters.
As d increases, the model becomes capable of representing increasingly complicated functions. In the limit, a sufficiently high-degree polynomial can interpolate all training points exactly. However, increasing the number of parameters also increases the variance of the estimator, making the model more sensitive to fluctuations in the training data.

Worked Examples

Concepts become clearer when applied to concrete examples. The book therefore includes examples illustrating how ideas appear in real modelling problems.

Worked Example

Suppose we fit polynomial regression models of degree 1, 3, and 15 to the same dataset. The linear model may miss curvature in the data and have high bias. The degree-15 model may fit almost every fluctuation and have high variance.

Python Implementations

Many interview processes include coding questions or discussions about implementation details. For this reason, full practical Python examples are included throughout the book using libraries such as NumPy, pandas, scikit-learn, PyTorch and statsmodels. These examples are simple in nature, but help understand the topic from a coding perspective. The notebook files can be downloaded directly from `github` here: https://github.com/coherentnoise.

```python
import numpy as np
from sklearn.linear_model import LinearRegression

X = np.array([[1], [2], [3], [4], [5]])
y = np.array([2, 4, 5, 4, 5])

model = LinearRegression()
model.fit(X, y)

print("Coefficient:", model.coef_)
print("Intercept:", model.intercept_)
```

Listing 1.1: Simple Python example

Follow-Up Questions

Interviewers frequently ask additional questions to test deeper understanding. Each section therefore includes examples of typical follow-up questions.

Follow-Up Interview Questions

- How does regularisation affect the bias–variance trade-off?
- Why does more flexible modelling usually increase variance?
- How does cross-validation help identify the right level of complexity?

Common Mistakes

We will highlight any common mistakes in this red textbox that students and interviewees make when trying to answer a particular question.

Common Mistake

A weak answer is to say only that bias is underfitting and variance is overfitting, without explaining why these occur or how the trade-off affects model selection.

1.8 How to Use This Book

Students preparing for interviews may read the chapters sequentially, gradually strengthening their understanding of key ideas in statistics, machine learning, and artificial intelligence.

Alternatively, readers may focus on specific areas relevant to their target roles, such as statistical modelling, machine learning algorithms, deep learning or time series analysis. Because each topic is organised around a specific interview question, the book can also be used as a reference when reviewing individual concepts.

1.9 Question Difficulty Levels

This book is organised around technical interview questions in data science, machine learning, and artificial intelligence. Each question is written at one of three levels:

Question Difficulty Levels

Core	**Example:** *What is logistic regression?* This is a core question because it tests a fundamental concept that appears frequently in data science and machine learning interviews. Most candidates are expected to answer it clearly and correctly.
Advanced	**Example:** *Why does bagging reduce variance?* This is an advanced question because it requires more than a definition. A strong answer usually involves reasoning about model instability, averaging, and the effect of correlation between estimators.
Deep / Research level	**Example:** *How does the bias–variance decomposition extend to ensemble models, and why does bagging reduce variance but not bias?* This is a deep question because it requires the candidate to connect multiple ideas, move beyond standard interview definitions, and reason carefully about the behaviour of ensembles at a more theoretical level.

These examples are not intended to define the difficulty levels rigidly, since the depth expected in an interview will always depend on the role. However, they provide a useful guide to the level of understanding each tag is meant to represent.

A good study strategy is to master the Core questions first, then move on to the Advanced and Deep questions. In a typical interview, these will start by assessing your core understanding of a topic, then ask more advanced questions before asking more open ended questions to assess your deeper understanding of the field.

1.10 A Final Note

Data science sits at the intersection of statistics, mathematics, and computer science. Successful practitioners combine theoretical understanding with practical problem-solving skills. The aim of this book is not simply to help students pass interviews. It is to help them develop the depth of understanding that allows them to **think like professional data scientists**.

By working through the questions in this book, readers will not only improve their interview performance but will also gain a deeper understanding of the mathematical and conceptual foundations of modern data science.

Part I

Modelling and Theory

The first part of this book lays the conceptual foundation for everything that follows. In machine learning interviews, many questions that appear practical on the surface are really testing whether you understand the underlying modelling ideas: what a model is assuming, what it is trying to learn, how flexibility affects behaviour, and why some methods generalise better than others. For that reason, this part focuses on the core principles that sit behind predictive modelling rather than on implementation details alone.

At its heart, modelling is about representing structure in data. A good model captures meaningful relationships between inputs and outputs without becoming so rigid that it misses important patterns or so flexible that it begins to fit noise. This part therefore introduces the language of function approximation, bias and variance, linear and non-linear modelling, regularisation, and the assumptions that make different modelling choices appropriate. These are the ideas that allow a candidate not just to name algorithms in an interview, but to explain why one modelling approach might succeed where another fails.

This part also develops the mathematical perspective that supports later chapters. Many interview questions become much easier once you understand how loss functions define learning objectives, how parameters are estimated, how complexity affects generalisation, and how optimisation interacts with model structure. The goal is not to turn every answer into a proof, but to give you the depth needed to move confidently between intuition, formal reasoning, and practical interpretation. In that sense, Part I provides the theoretical grammar of machine learning: it equips you with the concepts that make the rest of the subject coherent.

Chapter 2

The Bias–Variance trade-off

Introduction

One of the most fundamental ideas in machine learning is the bias–variance trade-off. This concept helps explain why models sometimes perform poorly on new data and why increasing model complexity does not always improve performance.

Understanding the bias–variance trade-off is essential for understanding many important topics in machine learning, including:

- model complexity
- overfitting and underfitting
- regularisation
- cross-validation
- model selection

For this reason, the bias–variance trade-off appears frequently in technical interviews for data science and machine learning roles.

Interview Question

Question 1: What is the bias–variance trade-off?

Difficulty Level

Tier: Core Question

2.0.1 Short Interview Answer

Short Interview Answer

The bias–variance trade-off describes the balance between two sources of prediction error in machine learning models.
Bias measures the error introduced by simplifying assumptions in the model, while variance measures how sensitive the model is to fluctuations in the training data.
Simple models typically have high bias and low variance, while complex models tend to have low bias but high variance. A good model achieves a balance between these two sources of error in order to generalise well to new data.

2.0.2 Intuition

To understand the bias–variance trade-off, consider the problem of fitting a model to data. If the model is too simple, it may fail to capture the underlying patterns in the data. This is known as **underfitting**. In this case the model makes systematic errors because it is too constrained to represent the true relationship. On the other hand, if the model is very flexible, it may fit the training data extremely well but also capture random noise in the dataset. This behaviour is known as **overfitting**. Such models perform well on the training data but poorly on unseen data.

The bias–variance trade-off explains this behaviour:

- Simple models tend to have **high bias** because they cannot represent complex patterns.
- Flexible models tend to have **high variance** because small changes in the training data can lead to very different models.

The goal of machine learning is therefore to find a model that balances these two effects.

Interview Tip

When answering this question in an interview, it is usually best to begin with the intuition (underfitting vs overfitting) before introducing the mathematical definition.

2.0.3 Mathematical Explanation

The bias–variance trade-off can be understood by analysing the expected prediction error of a model.

Mathematical Insight

The bias–variance decomposition arises by analysing the expected squared prediction error. This decomposition is known as the **bias–variance decomposition of expected prediction error**.
Suppose the true data generating process is

$$y = f(x) + \epsilon$$

where the noise term satisfies

$$\mathbb{E}[\epsilon] = 0, \qquad \mathrm{Var}(\epsilon) = \sigma^2.$$

If a model produces predictions $\hat{f}(x)$, then the expected squared prediction error can be written as

$$\mathbb{E}[(y - \hat{f}(x))^2] = \left(\mathbb{E}[\hat{f}(x)] - f(x)\right)^2 + \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2] + \sigma^2.$$

These three terms correspond respectively to

- squared bias
- variance
- irreducible noise

This decomposition explains why increasing model flexibility typically reduces bias but increases variance.

This decomposition shows that prediction error arises from multiple sources. Increasing model flexibility often reduces bias but increases variance.

Deep Dive

The irreducible noise term cannot be eliminated by any model because it arises from randomness in the data generating process. The goal of model selection is therefore to minimise the sum of bias and variance.

2.0.4 Visualising Bias and Variance

Another helpful way to understand bias and variance is through the **bullseye diagram**. In this visualisation, the centre of the target represents the true value of the quantity we are trying to predict, while each point represents the prediction produced by a model trained on a particular

dataset. The diagram in Figure 2.1 illustrates four possible situations:

- **Low bias, low variance.** Predictions are tightly clustered around the true value. The model is both accurate and stable.
- **High bias, low variance.** Predictions are tightly clustered but systematically displaced from the true value. The model consistently makes the same type of error, indicating that its assumptions are too restrictive.
- **Low bias, high variance.** Predictions are centred around the true value but are widely scattered. The model is flexible enough to capture the true relationship but is highly sensitive to fluctuations in the training data.
- **High bias, high variance.** Predictions are both widely dispersed and far from the true value. In this case the model is both inaccurate and unstable.

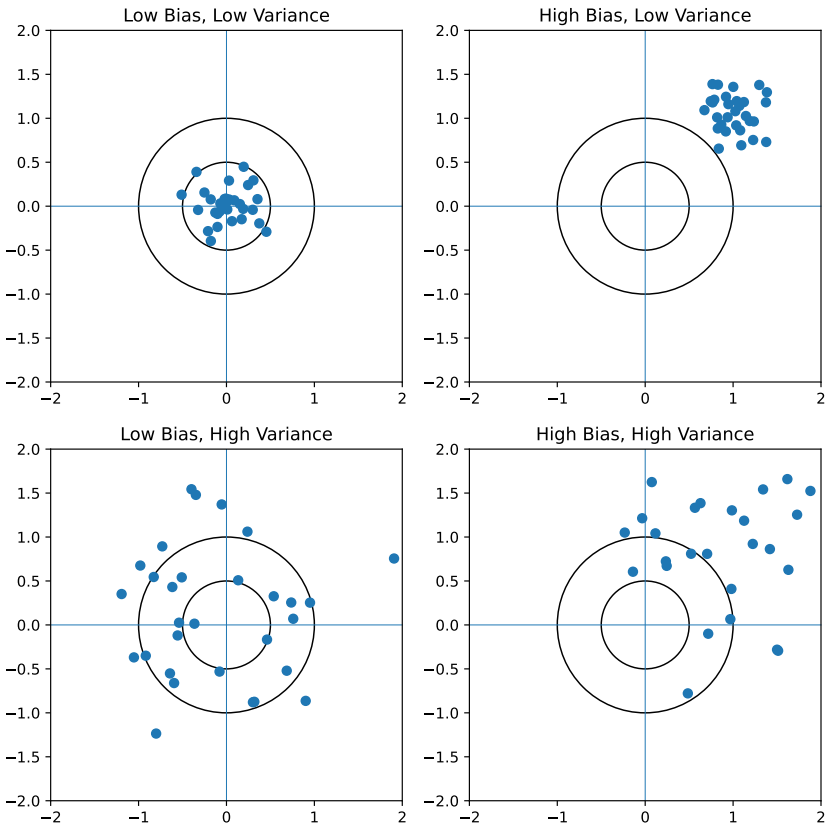


Figure 2.1: Illustration of the bias vs variance trade-off.

This visualisation highlights the two key components of prediction error:

- **Bias** measures how far the average prediction is from the true value.
- **Variance** measures how much predictions vary across different training datasets.

The goal of machine learning is to construct models that achieve both low bias and low variance.

2.0.5 Worked Example

Worked Example

Consider fitting polynomial regression models to the same dataset.

- A linear model may be too simple to capture curvature in the data. This leads to high bias.
- A high-degree polynomial may pass through nearly every training point, including noise. This leads to high variance.

The optimal model typically lies somewhere between these two extremes.

2.0.6 Python Demonstration

The following example illustrates the bias–variance trade-off by fitting polynomial models of increasing complexity.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

np.random.seed(0)

X = np.linspace(-3, 3, 30)
y = np.sin(X) + np.random.normal(0, 0.2, size=len(X))

X = X.reshape(-1,1)

degrees = [1, 3, 10]

for d in degrees:
    poly = PolynomialFeatures(degree=d)
    X_poly = poly.fit_transform(X)

    model = LinearRegression()
    model.fit(X_poly, y)

    X_test = np.linspace(-3,3,100).reshape(-1,1)
    X_test_poly = poly.transform(X_test)

    y_pred = model.predict(X_test_poly)

    plt.plot(X_test, y_pred, label=f"degree {d}")

plt.scatter(X, y)
```

```
plt.legend()
plt.title("Bias-Variance trade-off Example")
plt.show()
```

Listing 2.1: Illustrating the bias–variance trade-off with polynomial regression

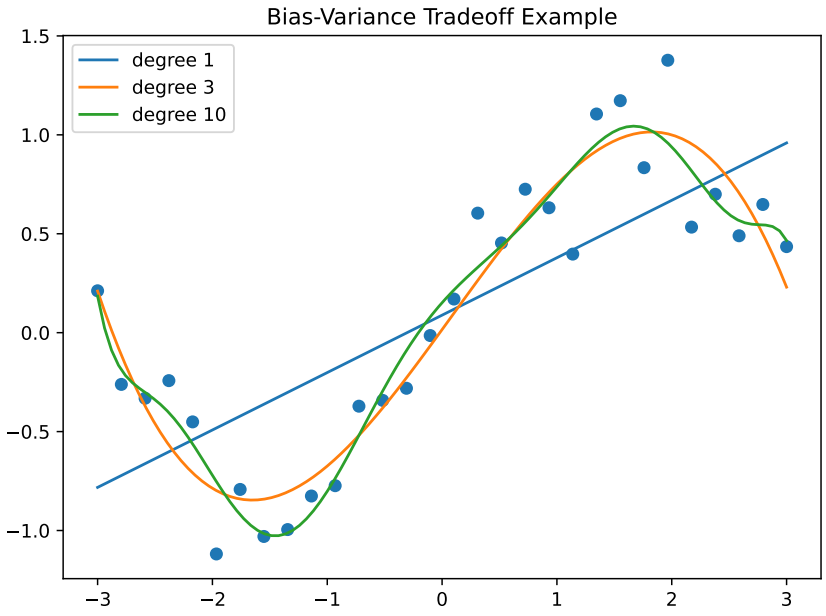


Figure 2.2: Illustration of the bias vs variance trade-off.

This experiment demonstrates how increasing model flexibility changes the fitted function. Low-degree models may fail to capture the structure of the data, while very high-degree models may follow noise in the dataset.

2.0.7 Visual Interpretation

The bias–variance trade-off is often illustrated using the relationship between model complexity and prediction error. Another common visualisation of the bias–variance trade-off shows how different components of prediction error change as model complexity increases.

- As model complexity increases, bias decreases.
- As model complexity increases, variance increases.
- The total prediction error typically follows a U-shaped curve.

The optimal model complexity occurs near the minimum of this curve.

2.0.8 How the Bias–Variance trade-off Appears in Practice

The bias–variance trade-off appears in many practical machine learning decisions, including:

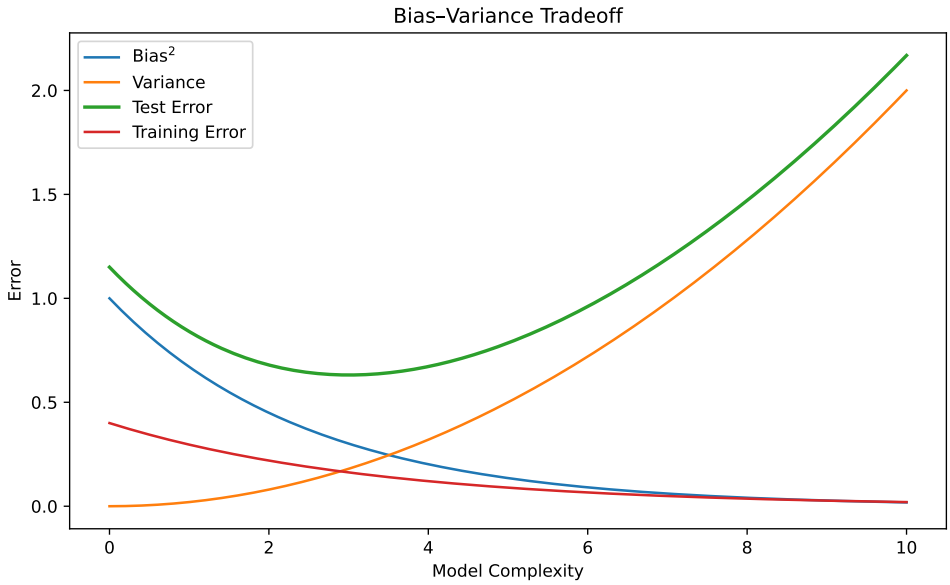


Figure 2.3: The bias–variance trade-off. As model complexity increases, bias decreases and variance increases. The optimal model occurs near the minimum of the test error curve.

- Choosing the depth of decision trees
- Selecting the number of parameters in neural networks
- Adjusting regularisation strength
- Determining the complexity of polynomial models

Many techniques in modern machine learning, including regularisation, ensemble learning, and cross-validation, can be understood as methods for managing the bias–variance trade-off.

2.0.9 Follow-Up Interview Questions

Follow-Up Interview Questions

- Why do flexible models typically have higher variance?
- How does regularisation affect the bias–variance trade-off?
- Why does increasing training data reduce variance?
- How does cross-validation help select the appropriate model complexity?
- How does ensemble learning reduce variance?

2.0.10 Common Mistakes

Common Mistake

Defining bias as underfitting and variance as overfitting without explaining why these phenomena occur.

Common Mistake

Failing to mention the mathematical decomposition of prediction error.

Common Mistake

Ignoring the role of irreducible noise in prediction error.

2.0.11 Summary

The bias–variance trade-off provides a framework for understanding why models generalise poorly when they are either too simple or too complex. Bias arises from overly restrictive modelling assumptions, while variance arises from a model's sensitivity to the particular training data it receives. As model complexity increases, bias typically decreases because the model becomes more flexible and better able to capture structure in the data, but variance usually increases because the model becomes more responsive to fluctuations and noise in the sample. The central challenge in model selection is therefore to balance these two sources of error so that the model is neither too simple nor too unstable. This trade-off plays a fundamental role throughout machine learning, influencing ideas such as regularisation, cross-validation, and ensemble learning.

Interview Question

Question 2: What is overfitting and underfitting in machine learning?

Difficulty Level

Tier: Core Question

2.0.12 Short Interview Answer

Short Interview Answer

Underfitting occurs when a model is too simple to capture the underlying structure of the data. As a result, it performs poorly on both the training data and new unseen data.
Overfitting occurs when a model becomes too complex and learns noise in the training dataset rather than the true underlying pattern. In this case the model performs extremely well on the training data but poorly on new data.
The goal of machine learning is therefore to find a model that is complex enough to capture meaningful patterns in the data but simple enough to generalise well to unseen observations.

2.0.13 Intuition

To understand underfitting and overfitting, imagine trying to model a relationship between two variables. If we attempt to fit a straight line to data that follows a curved pattern, the model will fail to capture the structure of the data. The predictions will therefore be systematically wrong. This situation is known as **underfitting**.

On the other hand, if we fit a very flexible model, such as a high-degree polynomial, the model may fit every training point extremely closely. While this may appear desirable, the model may also capture random noise in the dataset rather than the underlying signal. Such a model will perform very well on the training data but poorly on new unseen observations. This situation is known as **overfitting**. These two phenomena correspond closely to the bias–variance trade-off discussed earlier:

- Underfitting corresponds to **high bias**.
- Overfitting corresponds to **high variance**.

Interview Tip

In interviews, a strong explanation connects overfitting and underfitting to the bias–variance trade-off rather than describing them as isolated concepts.

2.0.14 Mathematical Perspective

Mathematical Insight

Suppose the true relationship between inputs and outputs is

$$y = f(x) + \epsilon$$

where ϵ represents random noise.
Let $\hat{f}(x)$ denote the model learned from training data.
The expected prediction error can be decomposed as

$$\mathbb{E}[(y - \hat{f}(x))^2] = \text{Bias}^2 + \text{Variance} + \text{Noise}.$$

Underfitting occurs when the bias term dominates the error because the model is too simple to approximate the true function $f(x)$.
Overfitting occurs when the variance term becomes large because the model adapts excessively to fluctuations in the training data.

Deep Dive

One way to detect overfitting is to compare training error and validation error.

- Underfitting: high training error and high validation error.
- Overfitting: low training error but high validation error.
- Good generalisation: both errors are relatively low and similar in magnitude.

These patterns are often visualised using learning curves.

2.0.15 Worked Example

Worked Example

Consider attempting to model the relationship

$$y = \sin(x) + \epsilon.$$

If we fit a simple linear model

$$y = ax + b,$$

the model will not capture the sinusoidal structure of the data. The predictions will therefore be inaccurate across the dataset. This is an example of underfitting.
If instead we fit a polynomial of degree 15, the model may pass extremely close to each training point. However, the resulting function may oscillate dramatically between points and produce poor predictions on new data. This is overfitting.

The optimal model lies somewhere between these two extremes.

2.0.16 Python Demonstration

The following example illustrates underfitting and overfitting by fitting polynomial models of increasing complexity.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

np.random.seed(1)

# Generate synthetic data
X = np.linspace(-3, 3, 30)
y = np.sin(X) + np.random.normal(0, 0.15, size=len(X))

X = X.reshape(-1,1)

# Test points
X_test = np.linspace(-3,3,200).reshape(-1,1)

degrees = [1, 4, 15]

plt.figure(figsize=(8,5))

for d in degrees:

    model = make_pipeline(
        PolynomialFeatures(degree=d),
        LinearRegression()
    )

    model.fit(X, y)
    y_pred = model.predict(X_test)

    plt.plot(X_test, y_pred, label=f"degree {d}")

plt.scatter(X, y, color="black", s=30)
plt.legend()
plt.title("Underfitting vs Overfitting")
plt.show()
```

Listing 2.2: Demonstrating underfitting and overfitting with polynomial regression

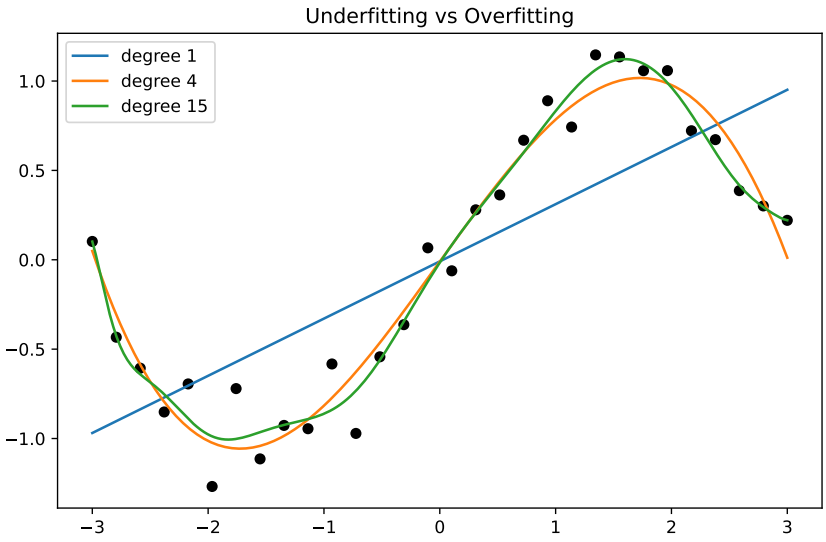


Figure 2.4: Demonstrating underfitting and overfitting with polynomial regression.

A low-degree polynomial fails to capture the structure of the data, while a very high-degree polynomial produces highly unstable predictions.

2.0.17 How Overfitting Appears in Practice

Overfitting can arise in many situations, including; models with too many parameters, small training datasets, noisy data, excessive training iterations. Common techniques used to prevent overfitting include **regularisation, cross-validation, early stopping, dropout in neural networks**, which will be covered in later chapters.

2.0.18 Follow-Up Interview Questions

Follow-Up Interview Questions

- How does regularisation help reduce overfitting?
- Why does increasing training data often reduce overfitting?
- What is the role of cross-validation in detecting overfitting?
- How does early stopping prevent overfitting in neural networks?

2.0.19 Common Mistakes

Common Mistake

Describing overfitting as simply "memorising the data" without explaining the relationship to model variance and model complexity.

Common Mistake

Failing to mention that underfitting results from overly restrictive model assumptions.

2.0.20 Summary

Underfitting and overfitting represent two common failure modes in machine learning models. The key points to remember in an interview are:

- Underfitting occurs when a model is too simple to capture the underlying data structure.
- Overfitting occurs when a model becomes too sensitive to the training data.
- These phenomena are closely related to the bias–variance trade-off.
- Effective model selection seeks a balance between these two extremes.

Interview Question

Question 3: What is model complexity in machine learning?

Difficulty Level

Tier: Advanced

2.0.21 Short Interview Answer

Short Interview Answer

Model complexity refers to the flexibility of a machine learning model and its ability to represent complex relationships in data.
Models with low complexity make strong assumptions about the structure of the data and may struggle to capture complicated patterns. Models with high complexity are more flexible and can represent a wider range of functions, but they are also more likely to overfit the training data.
Choosing an appropriate level of model complexity is therefore essential for achieving good generalisation.

2.0.22 Intuition

Model complexity can be thought of as the number of ways a model can adapt to data. For example:

- A simple linear regression model has relatively low complexity.
- A polynomial regression model with many terms is more complex.
- A deep neural network with millions of parameters has extremely high complexity.

More complex models can represent more intricate relationships between variables. However, this flexibility also means the model may fit noise in the training data rather than the underlying signal. As a result, increasing model complexity typically decreases bias but increases variance.

2.0.23 Mathematical Perspective

Mathematical Insight

Model complexity is often related to the number of parameters in a model.
Consider a polynomial regression model of degree d:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_d x^d.$$

This model contains $d + 1$ parameters.

As d increases, the model becomes capable of representing increasingly complicated functions. In the limit, a sufficiently high-degree polynomial can interpolate all training points exactly. However, increasing the number of parameters also increases the variance of the estimator, making the model more sensitive to fluctuations in the training data.

Deep Dive

In statistical learning theory, model complexity is often measured using concepts such as:

- the number of model parameters
- the Vapnik–Chervonenkis (VC) dimension
- the capacity of a hypothesis class

These measures quantify how flexible a model class is and how easily it can fit arbitrary datasets.

2.0.24 Worked Example

Worked Example

Suppose we attempt to approximate the function

$$y = \sin(x)$$

using polynomial regression.
A polynomial of degree 1 (a straight line) has very low complexity and will fail to capture the curvature of the function.
A polynomial of degree 5 may capture the general shape of the function quite well.
A polynomial of degree 20 has very high complexity and may pass through almost every training point exactly, including noise.
Thus increasing model complexity improves the ability to fit training data but can reduce generalisation performance.

2.0.25 Python Demonstration

The following example illustrates how increasing model complexity affects training error and test error.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

```
8
9  np.random.seed(123)
10
11 # Generate synthetic data
12 X = np.linspace(-3, 3, 80)
13 y_true = np.sin(X)
14 y = y_true + np.random.normal(0, 0.2, size=len(X))
15
16 X = X.reshape(-1, 1)
17
18 # Train/test split
19 X_train, X_test, y_train, y_test = train_test_split(
20     X, y, test_size=0.3, random_state=0
21 )
22
23 # Grid for smooth curves
24 X_plot = np.linspace(-3, 3, 300).reshape(-1, 1)
25
26 # Model degrees for visualization
27 degrees = [1, 4, 12]
28
29 # Degrees for error curve
30 degrees_full = range(1, 16)
31 train_errors = []
32 test_errors = []
33
34 for d in degrees_full:
35     model = make_pipeline(
36         PolynomialFeatures(d),
37         LinearRegression()
38     )
39     model.fit(X_train, y_train)
40
41     train_pred = model.predict(X_train)
42     test_pred = model.predict(X_test)
43
44     train_errors.append(mean_squared_error(y_train, train_pred))
45     test_errors.append(mean_squared_error(y_test, test_pred))
46
47 # Create side-by-side plots
48 fig, axes = plt.subplots(1, 2, figsize=(12,5))
49
50 # ----- Left: fitted models -----
51 axes[0].scatter(X_train, y_train, label="training data", alpha=0.7)
52 axes[0].scatter(X_test, y_test, label="test data", alpha=0.7)
53
54 for d in degrees:
55     model = make_pipeline(
```

```
        PolynomialFeatures(d),
        LinearRegression()
    )
    model.fit(X_train, y_train)
    y_plot = model.predict(X_plot)
    axes[0].plot(X_plot, y_plot, label=f"degree {d}")

axes[0].plot(X_plot, np.sin(X_plot), linestyle="--", label="true function")
axes[0].set_title("Model Fits with Different Complexity")
axes[0].set_xlabel("x")
axes[0].set_ylabel("y")
axes[0].legend()

# ----- Right: error vs complexity -----
axes[1].plot(degrees_full, train_errors, marker='o', label="training error"
    )
axes[1].plot(degrees_full, test_errors, marker='o', label="test error")

axes[1].set_title("Error vs Model Complexity")
axes[1].set_xlabel("Polynomial degree")
axes[1].set_ylabel("Mean squared error")
axes[1].legend()
plt.show()
```

Listing 2.3: Model complexity and polynomial regression

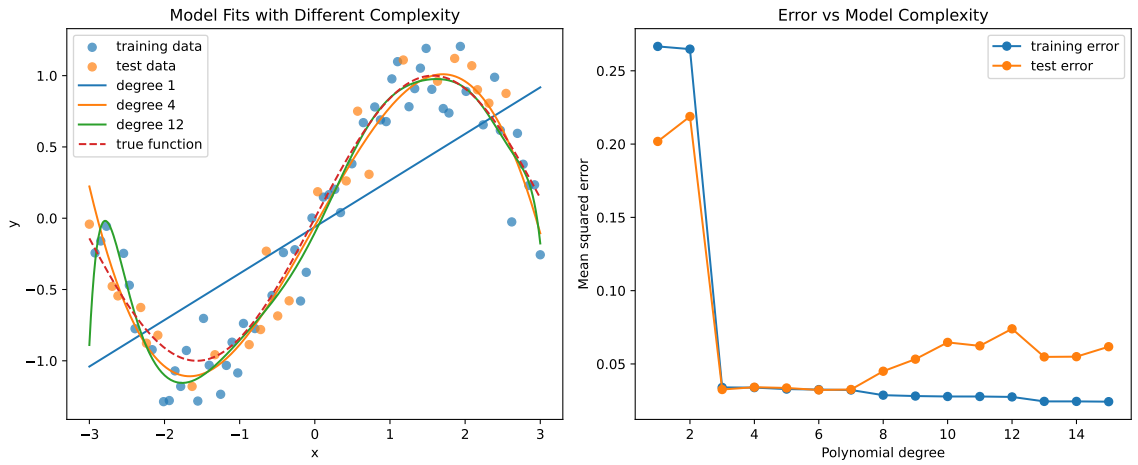


Figure 2.5: Illustration of model complexity and its effect on training and test error.

As the degree of the polynomial increases, the training error decreases steadily. However, the test error initially decreases and then begins to increase once the model becomes too complex.

2.0.26 How Model Complexity Appears in Practice

In real machine learning problems, model complexity can be controlled in several ways:

- limiting the depth of decision trees
- reducing the number of parameters in neural networks
- applying regularisation penalties
- early stopping during training

Many machine learning techniques can therefore be interpreted as methods for controlling model complexity.

2.0.27 Follow-Up Interview Questions

Follow-Up Interview Questions

- Why does increasing model complexity usually reduce bias?
- Why does increasing model complexity increase variance?
- How does regularisation control model complexity?
- How does cross-validation help determine the appropriate level of model complexity?

2.0.28 Common Mistakes

Common Mistake

Equating model complexity only with the number of parameters. While parameter count is important, other factors such as model architecture and hypothesis class also influence complexity.

Common Mistake

Failing to connect model complexity with the bias–variance trade-off.

2.0.29 Summary

Model complexity describes the flexibility of a machine learning model and its ability to represent different functions. The key points to remember in an interview include:

- Increasing model complexity reduces bias but increases variance.
- Very simple models may underfit the data.
- Highly flexible models may overfit the training data.
- Techniques such as regularisation and cross-validation help control model complexity.

Part II

Optimisation and Training

If Part I is about what models are, Part II is about how they are actually learned. In practice, even a well-chosen model is only useful if we can train it effectively, and many interview questions in machine learning are really questions about optimisation in disguise. Why does gradient descent work? Why do some networks train easily while others are unstable? What causes slow convergence, vanishing gradients, or overfitting during training? These are not secondary technical details; they are central to understanding real machine learning systems.

This part focuses on the dynamics of learning. It explains how loss functions, gradients, optimisation algorithms, initialisation strategies, activation functions, and regularisation techniques interact during training. Rather than treating training as a black box, the aim is to show how design choices influence whether optimisation is smooth or unstable, whether gradients remain informative across depth, and whether a model learns general structure or merely memorises the training set. In interviews, this is often where strong candidates distinguish themselves. This is because they can explain not only what method is used, but why training behaves the way it does.

Part II also connects theory to engineering practice. Training a model is rarely just a matter of running an optimiser until the loss decreases. One must think about learning rates, convergence behaviour, data splits, hyperparameter tuning, early stopping, and the signals provided by training and validation curves. These issues are especially important in modern machine learning, where model capacity is large and optimisation is often the bridge between elegant theory and messy real-world performance. By the end of this part, the reader should be able to speak clearly about how models learn, why training sometimes fails, and what practical steps can be taken to improve optimisation and generalisation.

Chapter 3

Loss Functions

Machine learning models learn by making mistakes. During training, a model makes predictions, compares those predictions to the true values, and then adjusts its parameters to reduce the error. The mathematical tool used to measure this error is called a **loss function**.

Loss functions are fundamental to the entire training process. They define what it means for a model to perform well or poorly, and they guide optimisation algorithms such as gradient descent toward better parameter values. Different machine learning problems require different types of loss functions. For example:

- Regression models commonly use **Mean Squared Error (MSE)**.
- Classification models often use **Cross-Entropy Loss**.
- Robust models may use **Absolute Error** or **Huber Loss**.

In practice, the loss function determines:

- how prediction errors are penalised
- how gradients are computed during training
- how sensitive the model is to outliers
- how optimisation behaves

In this chapter we explore what a loss function is, how loss differs from cost, common loss functions used in machine learning and how loss shapes the training process.

Interview Question

Question 4: What is a loss function in machine learning?

Difficulty Level

Tier: Core Question

3.0.1 Short Interview Answer

Short Interview Answer

A loss function is a mathematical function that measures how wrong a model's prediction is compared with the true target value.
For a single training example, it assigns a numerical penalty to the prediction error. Small errors produce small loss values, while large errors produce larger loss values.
During training, machine learning algorithms aim to minimise the loss so that the model's predictions become more accurate.

3.0.2 Intuition

A machine learning model learns by making predictions and then checking how far those predictions are from the correct answers. The loss function is the mechanism that tells the model how bad a particular prediction was.

For example, suppose a model is trying to predict house prices.

- If the true price is £300,000 and the model predicts £298,000, the error is small, so the loss should also be small.
- If the true price is £300,000 and the model predicts £180,000, the error is much larger, so the loss should be much larger.

In this sense, the loss function acts like a scoring rule for predictions. It tells the model whether it is performing well or poorly on each example. This is important because training algorithms such as gradient descent use the loss function to decide how to update the model parameters.

Interview Tip

In interviews, a strong answer should make clear that **loss usually refers to the error on a single example**. This distinction becomes important when comparing loss with cost or objective functions.

3.0.3 Mathematical Perspective

Mathematical Insight

A loss function is typically written as

$$L(y, \hat{y}),$$

where

- y is the true target value
- $\hat{y}$ is the prediction made by the model
- $L(y, \hat{y})$ is the penalty assigned to that prediction

A common example in regression is the squared error loss:

$$L(y, \hat{y}) = (y - \hat{y})^2.$$

This loss is small when the prediction is close to the true value and large when the prediction is far away.
For classification, a different loss function is often used, such as cross-entropy loss, because the outputs represent probabilities rather than continuous values.
More generally, the training process attempts to find model parameters θ that minimise the total loss over the training data.

This shows that the loss function is central to learning: it defines what the model is trying to optimise.

Deep Dive

Different loss functions penalise errors in different ways.
For example:

- Mean squared error penalises large errors very strongly because the error is squared.
- Mean absolute error penalises errors linearly and is therefore less sensitive to outliers.
- Cross-entropy loss is especially suitable for classification because it heavily penalises confident but incorrect predictions.

The choice of loss function therefore affects not only model evaluation but also the behaviour of the optimisation algorithm.

3.0.4 Worked Example

Worked Example

Suppose a regression model is predicting exam scores.
For one student, the true score is

$$y = 80$$

and the model predicts

$$\hat{y} = 75.$$

Using squared error loss, the loss is

$$L(y, \hat{y}) = (80 - 75)^2 = 25.$$

For another student, suppose the true score is 80 but the model predicts 60. Then

$$L(y, \hat{y}) = (80 - 60)^2 = 400.$$

The second prediction receives a much larger penalty because it is much further from the true value.

3.0.5 Python Demonstration

The following example computes squared error loss for a small set of predictions.

```python
import numpy as np

# True values
y_true = np.array([80, 65, 90, 72])

# Model predictions
y_pred = np.array([75, 70, 84, 60])

# Squared error loss for each example
loss = (y_true - y_pred) ** 2

print("Loss for each example:")
print(loss)
```

Listing 3.1: Computing squared error loss for individual predictions

This example shows that each prediction receives its own loss value depending on how accurate it is.

3.0.6 How Loss Functions Appear in Practice

Loss functions appear throughout machine learning because they define what the model is trying to learn. For example:

- In linear regression, models often minimise mean squared error.
- In logistic regression, models often minimise cross-entropy loss.
- In robust regression, absolute error or Huber loss may be used to reduce sensitivity to outliers.

Choosing an appropriate loss function is therefore an important part of model design.

3.0.7 Follow-Up Interview Questions

Follow-Up Interview Questions

- What is the difference between a loss function and a cost function?
- Why is cross-entropy loss preferred for classification?
- How does the choice of loss function affect optimisation?

3.0.8 Common Mistakes

Common Mistake

Describing a loss function only as an evaluation metric without explaining that it is also the quantity minimised during training.

Common Mistake

Failing to distinguish between the loss for a single example and the overall cost across a dataset.

3.0.9 Summary

A loss function measures how wrong a model's prediction is relative to the true target value by assigning a numerical penalty to prediction error. Small mistakes produce small loss values, while larger mistakes produce larger penalties, so the loss function gives the training algorithm a precise objective to minimise. In practice, learning consists of adjusting the model parameters to reduce this loss and therefore improve predictions. In an interview, the most important point to remember is that the loss function defines what the model is trying to optimise, and that the choice of loss depends on the task: squared error is common in regression, while cross-entropy is standard in classification.

Interview Question

Question 5: Why is mean squared error used in regression?

Difficulty Level

Tier: Core Question

3.0.10 Short Interview Answer

Short Interview Answer

Mean squared error (MSE) is commonly used in regression because it penalises large prediction errors more strongly than small ones and produces a smooth, differentiable objective that is easy to optimise using gradient-based methods.
It also has strong statistical justification: when the noise in the data is Gaussian, minimising mean squared error corresponds to the maximum likelihood estimate of the model parameters.

3.0.11 Intuition

In regression problems, the model predicts a continuous numerical value. To train the model, we need a way to measure how far each prediction is from the true value.

Mean squared error measures this by squaring the difference between the prediction and the true value.

$$L(y, \hat{y}) = (y - \hat{y})^2$$

Squaring the error has two important effects.

- Large errors are penalised much more strongly than small errors.
- Positive and negative errors do not cancel each other out.

For example, predicting 90 instead of 100 produces a smaller penalty than predicting 50 instead of 100. This behaviour encourages the model to avoid large mistakes, which often improves predictive accuracy.

Interview Tip

In interviews, it is useful to mention both the **practical optimisation reasons** and the **statistical justification** for using mean squared error.

3.0.12 Mathematical Perspective

Mathematical Insight

For a dataset with n observations, the mean squared error cost function is

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

This function has several useful properties.

- It is **continuous and differentiable**, which makes it suitable for gradient-based optimisation.
- The squared term produces a **convex objective** for linear regression, ensuring a unique global minimum.
- The gradient of the loss grows linearly with the error magnitude, which strongly penalises large prediction errors.

Because of these properties, optimisation algorithms such as gradient descent can efficiently minimise the mean squared error.

Deep Dive

Mean squared error also has an important statistical interpretation.
If the data generating process is

$$y = f(x) + \epsilon$$

and the noise term satisfies

$$\epsilon \sim \mathcal{N}(0, \sigma^2),$$

then minimising the sum of squared errors is equivalent to finding the maximum likelihood estimate of the model parameters.
This result provides a strong theoretical justification for the use of mean squared error in regression models.

3.0.13 Worked Example

Worked Example

Suppose a model predicts the following house prices (in hundreds of thousands of pounds).

True Price	Prediction	Squared Error
3.0	2.8	0.04
4.5	4.0	0.25
2.0	2.5	0.25

The mean squared error is

$$\text{MSE} = \frac{0.04 + 0.25 + 0.25}{3} = 0.18.$$

This value summarises the average squared prediction error of the model.

3.0.14 Python Demonstration

The following example computes the mean squared error for a simple regression prediction.

```python
import numpy as np

# True values
y_true = np.array([3.0, 4.5, 2.0])

# Model predictions
y_pred = np.array([2.8, 4.0, 2.5])

# Mean squared error
mse = np.mean((y_true - y_pred)**2)

print("Mean Squared Error:", mse)
```

Listing 3.2: Computing mean squared error

3.0.15 How Mean Squared Error Appears in Practice

Mean squared error is widely used across regression models, including:

- linear regression
- polynomial regression
- neural network regression models
- many gradient boosting algorithms

Although other regression losses exist (such as absolute error or Huber loss), mean squared error remains the most common choice due to its mathematical simplicity and strong theoretical foundations.

3.0.16 Follow-Up Interview Questions

Follow-Up Interview Questions

- Why does squaring the error penalise large mistakes more strongly?
- Why is mean squared error sensitive to outliers?
- Why does mean squared error correspond to maximum likelihood under Gaussian noise?
- What are alternatives to mean squared error for robust regression?

3.0.17 Common Mistakes

Common Mistake

Explaining mean squared error purely as a metric without mentioning its optimisation advantages or statistical interpretation.

Common Mistake

Forgetting that squaring the error makes the loss sensitive to large outliers.

3.0.18 Summary

Mean squared error is widely used in regression because it measures the average squared difference between predictions and true values, giving a loss that is both mathematically convenient and strongly sensitive to large mistakes. By squaring the error, it penalises large deviations much more heavily than small ones, which makes it useful when large prediction errors are especially undesirable. It is also smooth and differentiable, so it works well with gradient-based optimisation methods. In an interview, the key point to remember is that MSE is popular not just because it measures error, but because it provides a clean optimisation objective and has a strong probabilistic interpretation: under Gaussian noise assumptions, minimising MSE is equivalent to maximum likelihood estimation.

Interview Question

Question 6: Why is mean squared error equivalent to maximum likelihood under Gaussian noise?

Difficulty Level

Tier: Advanced

3.0.19 Short Interview Answer

Short Interview Answer

Mean squared error arises naturally from maximum likelihood estimation when the noise in the data is assumed to follow a Gaussian distribution.
If the observation model is

$$y = f(x) + \epsilon$$

with

$$\epsilon \sim \mathcal{N}(0, \sigma^2),$$

then maximising the likelihood of the data is equivalent to minimising the sum of squared prediction errors. This leads directly to the mean squared error objective used in regression.

3.0.20 Intuition

In many regression problems we assume that observations contain some random noise. For example, when predicting house prices, the true relationship between features and price might be

$$y = f(x),$$

but the actual observed price may differ slightly due to measurement errors, market fluctuations, or other unobserved factors. This randomness is often modelled using Gaussian noise:

$$y = f(x) + \epsilon,$$

where the noise term ϵ follows a normal distribution.

Under this assumption, predictions that are close to the true value are much more likely than predictions that are far away. When we compute the likelihood of the data under this model, the resulting optimisation problem turns out to be exactly the same as minimising squared

prediction errors.

Interview Tip

In interviews, a strong explanation begins with the Gaussian noise assumption and then shows how the likelihood leads to a squared error objective.

3.0.21 Mathematical Perspective

Mathematical Insight

Suppose the model predicts

$$\hat{y}_i = f(x_i; \theta),$$

and the observation model is

$$y_i = f(x_i; \theta) + \epsilon_i,$$

where

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2).$$

Under this assumption, the probability density of observing y_i is

$$p(y_i \mid x_i, \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \hat{y}_i)^2}{2\sigma^2}\right).$$

For a dataset with n observations, the likelihood is

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} p(y_i \mid x_i, \theta).$$

Taking the logarithm gives the log-likelihood

$$\log \mathcal{L}(\theta) = -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + C,$$

where C is a constant.
Maximising this log-likelihood is therefore equivalent to minimising

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

which is the sum of squared errors.

This derivation explains why squared error loss appears naturally in regression models with Gaussian noise.

Deep Dive

The factor $1/(2\sigma^2)$ does not affect the location of the optimum because it is a constant scaling factor.
For this reason, optimisation algorithms typically minimise the simpler objective

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

or its average, which is the mean squared error.
Thus mean squared error can be interpreted as the negative log-likelihood of a Gaussian noise model (up to a constant factor).

3.0.22 Worked Example

Worked Example

Suppose a regression model predicts exam scores.
The predicted score for a student is

$$\hat{y} = 75,$$

while the observed score is

$$y = 80.$$

If the noise is Gaussian with variance σ^2, the likelihood of observing this value decreases as the squared difference

$$(y - \hat{y})^2$$

becomes larger.
Predictions that minimise squared error therefore correspond to the parameter values that maximise the likelihood of the observed data.

3.0.23 Python Demonstration

The following example illustrates the relationship between squared error and the Gaussian likelihood.

```
import numpy as np

# observed value
y = 80

# predicted value
```

```
y_hat = 75

# assumed noise variance
sigma = 5

# squared error
squared_error = (y - y_hat)**2

# Gaussian likelihood
likelihood = np.exp(-squared_error / (2 * sigma**2))

print("Squared error:", squared_error)
print("Gaussian likelihood (unnormalised):", likelihood)
```

Listing 3.3: Gaussian likelihood and squared error

As the squared error increases, the likelihood decreases, which explains why minimising squared error corresponds to maximising likelihood.

3.0.24 How This Appears in Practice

This result explains why mean squared error is the standard loss function for many regression models, including:

- linear regression
- neural network regression models
- Gaussian process regression

In each case, the squared error objective can be interpreted as the negative log-likelihood of a Gaussian noise model.

3.0.25 Follow-Up Interview Questions

Follow-Up Interview Questions

- What loss function arises if the noise follows a Laplace distribution?
- Why is mean absolute error more robust to outliers?
- How does the Gaussian noise assumption influence regression modelling?
- Why is the log-likelihood used instead of the likelihood during optimisation?

3.0.26 Common Mistakes

Common Mistake

Stating that MSE corresponds to maximum likelihood without explaining the Gaussian noise assumption.

Common Mistake

Forgetting that the squared error arises from the exponential term in the Gaussian probability density function.

3.0.27 Summary

Mean squared error arises naturally from maximum likelihood estimation when the noise in the data follows a Gaussian distribution. The key points to remember in an interview setting are:

- Regression models often assume additive Gaussian noise.
- The Gaussian likelihood contains a squared error term in its exponent.
- Maximising the log-likelihood is equivalent to minimising squared prediction error.
- Mean squared error therefore has a strong statistical justification.

Interview Question

Question 7: Why is cross-entropy used for classification?

Difficulty Level

Tier: Core Question

3.0.28 Short Interview Answer

Short Interview Answer

Cross-entropy is used for classification because it measures the difference between the predicted probability distribution produced by the model and the true class distribution. It strongly penalises confident but incorrect predictions and has a natural probabilistic interpretation. In fact, minimising cross-entropy corresponds to maximising the likelihood of the observed class labels under the model.

3.0.29 Intuition

In classification problems, the goal of the model is not simply to predict a number but to estimate the probability that an input belongs to each class. For example, a binary classifier might produce the following prediction:

$$P(y = 1 \mid x) = 0.9.$$

If the true label is $y = 1$, this prediction is very good. If the true label is $y = 0$, however, the model was highly confident and completely wrong.

A good loss function for classification should therefore satisfy two important properties:

- Correct predictions with high confidence should produce very small loss.
- Incorrect predictions with high confidence should produce very large loss.

Cross-entropy satisfies exactly these requirements. As the model assigns lower probability to the correct class, the loss increases rapidly.

Interview Tip

In interviews, it is helpful to explain that cross-entropy evaluates how well the predicted probabilities match the true labels.

3.0.30 Mathematical Perspective

Mathematical Insight

For binary classification, the cross-entropy loss for a single observation is

$$L(y, \hat{p}) = -\big[y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})\big],$$

where

- $y \in \{0, 1\}$ is the true label
- $\hat{p} = P(y = 1 \mid x)$ is the predicted probability

If the true class is $y = 1$, the loss simplifies to

$$L = -\log(\hat{p}).$$

If the model assigns high probability to the correct class, the loss is small. If the probability is close to zero, the loss becomes very large.

This behaviour encourages the model to assign high probability to the correct class.

Deep Dive

Cross-entropy also arises naturally from maximum likelihood estimation.
If a classification model predicts class probabilities $\hat{p}$, we can interpret these probabilities as parameters of a Bernoulli distribution.
Maximising the likelihood of the observed labels under this distribution leads directly to the cross-entropy objective.
Thus minimising cross-entropy is equivalent to maximising the likelihood of the training labels under the model.

3.0.31 Worked Example

Worked Example

Suppose a binary classifier produces the following predicted probabilities for the positive class.

True Label	Predicted Probability	Cross-Entropy Loss
1	0.9	$-\log(0.9) = 0.105$
1	0.6	$-\log(0.6) = 0.511$
1	0.1	$-\log(0.1) = 2.303$

As the predicted probability for the correct class decreases, the loss increases rapidly.

3.0.32 Python Demonstration

The following example computes cross-entropy loss for a few binary classification predictions.

```python
import numpy as np

# true labels
y_true = np.array([1, 1, 0])

# predicted probabilities
p = np.array([0.9, 0.6, 0.2])

# binary cross-entropy loss
loss = - (y_true*np.log(p) + (1-y_true)*np.log(1-p))

print("Cross-entropy loss for each example:")
print(loss)
```

Listing 3.4: Computing cross-entropy loss

This example illustrates how cross-entropy increases rapidly when the predicted probability of the correct class becomes small.

3.0.33 How Cross-Entropy Appears in Practice

Cross-entropy is widely used for training classification models, including:

- logistic regression
- neural network classifiers
- softmax classifiers for multi-class problems
- many deep learning models

Because it corresponds to maximum likelihood estimation for probabilistic classifiers, it provides both a principled statistical objective and a practical training loss.

3.0.34 Follow-Up Interview Questions

Follow-Up Interview Questions

- Why does cross-entropy heavily penalise confident incorrect predictions?
- How does cross-entropy relate to maximum likelihood estimation?
- What is the difference between cross-entropy and log-loss?
- How is cross-entropy extended to multi-class classification?

3.0.35 Common Mistakes

Common Mistake

Explaining cross-entropy purely as a formula without mentioning that it measures the difference between predicted and true probability distributions.

Common Mistake

Failing to explain that cross-entropy corresponds to maximum likelihood estimation for probabilistic classifiers.

3.0.36 Summary

Cross-entropy is widely used in classification because it measures how well predicted probabilities match the true class labels. Key ideas to remember for a technical interview include:

- Cross-entropy evaluates the difference between predicted and true class probabilities.
- Incorrect predictions with high confidence produce large loss values.
- Minimising cross-entropy corresponds to maximising the likelihood of the observed labels.
- It is the standard loss function for many classification models.

Interview Question

Question 8: Why is cross-entropy always non-negative?

Difficulty Level

Tier: Advanced

3.0.37 Short Interview Answer

Short Interview Answer

Cross-entropy is always non-negative because it is defined as the negative logarithm of probabilities, and probabilities lie between 0 and 1. Since the logarithm of a number between 0 and 1 is negative, the negative log term becomes non-negative.
More fundamentally, cross-entropy measures the expected number of bits required to encode data drawn from the true distribution using a predicted distribution, and this quantity cannot be negative.

3.0.38 Intuition

Cross-entropy measures how well one probability distribution matches another. In classification problems, we compare:

- the **true distribution** over classes
- the **predicted distribution** produced by the model

If the model assigns high probability to the correct class, the loss is small. If it assigns very low probability to the correct class, the loss becomes large. The loss is computed using the negative logarithm of the predicted probability of the correct class. For example:

$$L = -\log(\hat{p})$$

Since predicted probabilities satisfy

$$0 < \hat{p} \leq 1,$$

their logarithm is always less than or equal to zero.

Therefore

$$-\log(\hat{p}) \geq 0.$$

This means the cross-entropy loss can never be negative.

Interview Tip

A strong interview answer can mention both the **logarithm property** and the **information-theoretic interpretation** of cross-entropy.

3.0.39 Mathematical Perspective

Mathematical Insight

For discrete distributions p and q, the cross-entropy is defined as

$$H(p, q) = -\sum_i p_i \log q_i.$$

Here

- p_i is the true probability of class i
- q_i is the predicted probability assigned by the model

Since

$$0 < q_i \leq 1,$$

we have

$$\log q_i \leq 0.$$

Because $p_i \geq 0$, every term in the sum satisfies

$$-p_i \log q_i \geq 0.$$

Therefore

$$H(p, q) \geq 0.$$

This property guarantees that cross-entropy loss values are always non-negative.

Deep Dive

Cross-entropy can also be expressed as

$$H(p, q) = H(p) + D_{\mathrm{KL}}(p \| q),$$

where

- $H(p)$ is the entropy of the true distribution

- $D_{\mathrm{KL}}(p\|q)$ is the Kullback–Leibler divergence

Since KL divergence is always non-negative, cross-entropy must also be non-negative. This decomposition provides a deeper information-theoretic explanation of the loss.

3.0.40 Worked Example

Worked Example

Suppose a binary classifier predicts

$$\hat{p} = 0.8$$

for the correct class.
The cross-entropy loss is

$$L = -\log(0.8) \approx 0.223.$$

If the model predicts

$$\hat{p} = 0.1,$$

the loss becomes

$$L = -\log(0.1) \approx 2.303.$$

In both cases the loss is positive, and it increases as the predicted probability of the correct class decreases.

3.0.41 Python Demonstration

The following example illustrates that cross-entropy loss is always non-negative.

```python
import numpy as np

# predicted probabilities for correct class
p = np.array([0.9, 0.7, 0.4, 0.1])

# cross-entropy loss
loss = -np.log(p)

print("Loss values:", loss)
```

Listing 3.5: Cross-entropy loss is always non-negative

All computed loss values are non-negative.

3.0.42 How This Appears in Practice

Because cross-entropy is always non-negative, a lower value indicates better predictions. For example:

- perfect predictions produce loss close to 0
- uncertain predictions produce moderate loss
- confident incorrect predictions produce very large loss

This property makes cross-entropy particularly effective for training classification models.

3.0.43 Follow-Up Interview Questions

Follow-Up Interview Questions

- What is the relationship between cross-entropy and KL divergence?
- Why does cross-entropy penalise confident incorrect predictions so strongly?
- How does cross-entropy behave when the predicted probability approaches zero?
- Why is cross-entropy preferred over squared error for classification?

3.0.44 Common Mistakes

Common Mistake

Claiming that cross-entropy can become negative when probabilities are large. Since probabilities are at most 1, the logarithm is never positive, so the loss cannot be negative.

Common Mistake

Ignoring the information-theoretic interpretation of cross-entropy and treating it purely as a formula.

3.0.45 Summary

Cross-entropy is always non-negative because it is built from the negative logarithm of probabilities. Since probabilities lie between 0 and 1, their logarithms are always non-positive, and taking the negative logarithm therefore produces a value that is never negative. This gives cross-entropy the natural interpretation of a loss: better probability assignments lead to smaller values, while assigning low probability to the true outcome produces a larger penalty. Cross-entropy can also be understood through its relationship with KL divergence, which provides another reason it is non-negative and helps connect it to a broader information-theoretic view of learning.

Interview Question

Question 9: What does minimising KL divergence mean for a predictive model?

Difficulty Level

Tier: Advanced

3.0.46 Short Interview Answer

Short Interview Answer

Minimising KL divergence means adjusting the model so that its predicted probability distribution becomes as close as possible to the true data distribution.
In other words, the model learns to produce probabilities that match the real distribution of labels in the data. In many machine learning algorithms, minimising cross-entropy is equivalent to minimising the KL divergence between the true distribution and the model's predicted distribution.

3.0.47 Intuition

In probabilistic machine learning, models do not simply produce predictions. Instead, they estimate probability distributions. For example, a classifier may predict

$$P(y = 0 \mid x) = 0.2, \quad P(y = 1 \mid x) = 0.8.$$

Ideally, these predicted probabilities should match the true underlying distribution of the data.

KL divergence measures how different two probability distributions are. Specifically, it measures how much information is lost when we use one distribution to approximate another. If a model minimises KL divergence during training, it is effectively learning a probability distribution that closely approximates the true distribution of the data.

Interview Tip

A clear interview explanation emphasises that KL divergence measures the difference between the **true data distribution** and the **model's predicted distribution**.

3.0.48 Mathematical Perspective

Mathematical Insight

The KL divergence between two discrete distributions p and q is defined as

$$D_{\mathrm{KL}}(p\|q) = \sum_i p_i \log \frac{p_i}{q_i}.$$

Here

- p represents the true data distribution
- q represents the model's predicted distribution

KL divergence measures the expected log difference between the two distributions.
It has two important properties:

- $D_{\mathrm{KL}}(p\|q) \geq 0$
- $D_{\mathrm{KL}}(p\|q) = 0$ only when $p = q$

Thus minimising KL divergence forces the predicted distribution q to become closer to the true distribution p.

Deep Dive

KL divergence is closely related to cross-entropy.
Recall that cross-entropy can be written as

$$H(p, q) = H(p) + D_{\mathrm{KL}}(p\|q).$$

The entropy $H(p)$ depends only on the true distribution and does not depend on the model. Therefore, minimising cross-entropy during training is equivalent to minimising the KL divergence between the true distribution and the predicted distribution.
This explains why cross-entropy loss is widely used for training probabilistic classifiers.

3.0.49 Worked Example

Worked Example

Suppose the true distribution of a binary label is

$$p = (0.7, 0.3).$$

A model predicts

$$q = (0.6, 0.4).$$

The KL divergence is

$$D_{\mathrm{KL}}(p\|q) = 0.7 \log \frac{0.7}{0.6} + 0.3 \log \frac{0.3}{0.4}.$$

If the model instead predicts

$$q = (0.7, 0.3),$$

then the KL divergence becomes zero, indicating that the predicted distribution perfectly matches the true distribution.

3.0.50 Python Demonstration

The following example computes the KL divergence between two probability distributions.

```python
import numpy as np

# true distribution
p = np.array([0.7, 0.3])

# predicted distribution
q = np.array([0.6, 0.4])

kl_div = np.sum(p * np.log(p / q))

print("KL divergence:", kl_div)
```

Listing 3.6: Computing KL divergence

The KL divergence decreases as the predicted distribution becomes closer to the true distribution.

3.0.51 How KL Divergence Appears in Practice

KL divergence plays an important role in many machine learning methods, including:

- probabilistic classification models
- variational inference
- variational autoencoders
- reinforcement learning algorithms

In each case, minimising KL divergence encourages the model to produce probability distributions that approximate the true underlying data distribution.

3.0.52 Follow-Up Interview Questions

Follow-Up Interview Questions

- Why is KL divergence not symmetric?
- What is the difference between forward KL and reverse KL?
- How does KL divergence relate to maximum likelihood estimation?
- Why is KL divergence widely used in variational inference?

3.0.53 Common Mistakes

Common Mistake

Describing KL divergence as a distance metric. KL divergence measures distributional difference but it is not symmetric and does not satisfy the triangle inequality.

Common Mistake

Failing to connect KL divergence with cross-entropy and maximum likelihood training.

3.0.54 Summary

KL divergence measures the difference between two probability distributions. The key ideas to remember for a tachnical interview include:

- KL divergence quantifies how well one distribution approximates another.
- It is always non-negative and equals zero only when the distributions match.
- Minimising KL divergence encourages the model to match the true data distribution.
- Minimising cross-entropy during training is equivalent to minimising KL divergence.

Interview Question

Question 10: What is log-loss and how is it related to cross-entropy?

Difficulty Level

Tier: Core Question

3.0.55 Short Interview Answer

Short Interview Answer

Log-loss is a loss function used in classification that measures how well predicted probabilities match the true class labels. It penalises incorrect predictions based on the negative logarithm of the predicted probability assigned to the correct class.
Log-loss is mathematically equivalent to cross-entropy when the true labels are represented as one-hot encoded probability distributions.

3.0.56 Intuition

In classification problems, models typically output probabilities for each class. For example, a binary classifier might produce the prediction

$$P(y = 1 \mid x) = 0.8.$$

If the true label is $y = 1$, this prediction is good because the model assigned high probability to the correct class. If the model instead predicts

$$P(y = 1 \mid x) = 0.1,$$

then the prediction is very poor because the model was confident in the wrong answer. Log-loss measures this error by taking the negative logarithm of the probability assigned to the correct class:

$$L = -\log(\hat{p}_{\text{correct}}).$$

This creates two desirable properties:

- Correct predictions with high probability produce very small loss.
- Incorrect predictions with low probability produce very large loss.

Interview Tip

In interviews, a strong answer notes that log-loss is simply the practical form of cross-entropy used when training classifiers.

3.0.57 Mathematical Perspective

Mathematical Insight

For binary classification, log-loss is defined as

$$L(y, \hat{p}) = -\big[y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})\big],$$

where

- y is the true label (0 or 1)
- $\hat{p}$ is the predicted probability for the positive class

For multi-class classification, log-loss becomes

$$L = -\sum_{i} y_i \log(\hat{p}_i),$$

where y_i is a one-hot encoded label.
This expression is exactly the same as the cross-entropy between the true distribution p and the predicted distribution q:

$$H(p, q) = -\sum_{i} p_i \log q_i.$$

Thus log-loss and cross-entropy are mathematically equivalent when the true labels are represented as probability distributions.

Deep Dive

Log-loss can also be interpreted from a probabilistic perspective.
If a classifier outputs predicted probabilities $\hat{p}$, we can treat these probabilities as parameters of a Bernoulli or categorical distribution.
Maximising the likelihood of the observed labels leads directly to the log-loss objective. Therefore minimising log-loss is equivalent to performing maximum likelihood estimation for probabilistic classifiers.

3.0.58 Worked Example

Worked Example

Suppose a classifier predicts the probability of the positive class.

True Label	Predicted Probability	Log-Loss
1	0.9	$-\log(0.9) = 0.105$
1	0.6	$-\log(0.6) = 0.511$
1	0.1	$-\log(0.1) = 2.303$

The loss increases rapidly as the predicted probability for the correct class becomes smaller.

3.0.59 Python Demonstration

```python
import numpy as np

# true labels
y = np.array([1, 0, 1])

# predicted probabilities
p = np.array([0.9, 0.2, 0.6])

loss = -(y*np.log(p) + (1-y)*np.log(1-p))

print("Log-loss values:", loss)
```

Listing 3.7: Computing log-loss

3.0.60 Follow-Up Interview Questions

Follow-Up Interview Questions

- Why does log-loss penalise confident incorrect predictions so strongly?
- How does log-loss relate to maximum likelihood estimation?
- Why is log-loss preferred over squared error for classification?
- How does log-loss behave when predicted probabilities approach zero?

3.0.61 Common Mistakes

Common Mistake

Treating log-loss and cross-entropy as completely different loss functions when they are mathematically equivalent in classification settings.

Common Mistake

Failing to explain that log-loss evaluates predicted probabilities rather than raw class predictions.

3.0.62 Summary

Log-loss is a classification loss function that penalises incorrect probabilistic predictions. The key ideas to remember in an interview setting include:

- Log-loss measures the negative log probability assigned to the correct class.
- It heavily penalises confident incorrect predictions.
- Log-loss is mathematically equivalent to cross-entropy when labels are one-hot encoded.
- Minimising log-loss corresponds to maximum likelihood estimation for probabilistic classifiers.

Interview Question

Question 11: Why are log probabilities used in machine learning optimisation?

Difficulty Level

Tier: Advanced

3.0.63 Short Interview Answer

Short Interview Answer

Log probabilities are used in machine learning optimisation because they transform products of probabilities into sums, improve numerical stability, and produce gradients that are easier to optimise.

Taking logarithms also converts likelihood maximisation into log-likelihood maximisation, which simplifies many optimisation problems.

3.0.64 Intuition

Many machine learning models are probabilistic. This means that the likelihood of a dataset is often expressed as a product of probabilities:

$$P(D|\theta) = \prod_{i=1}^{n} P(y_i|x_i, \theta).$$

When the dataset is large, multiplying many probabilities together can produce extremely small numbers that are difficult for computers to represent accurately. Taking the logarithm solves this problem directly due to the fact that logarithms convert products into sums,

$$\log P(D|\theta) = \sum_{i=1}^{n} \log P(y_i|x_i, \theta).$$

This transformation makes optimisation much more stable and computationally convenient.

Interview Tip

A strong interview answer highlights three reasons for using log probabilities: numerical stability, simpler optimisation, and easier gradient computation.

3.0.65 Mathematical Perspective

Mathematical Insight

Suppose the likelihood of a dataset is

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} p(y_i|x_i, \theta).$$

Taking the logarithm gives the log-likelihood

$$\log \mathcal{L}(\theta) = \sum_{i=1}^{n} \log p(y_i|x_i, \theta).$$

Because the logarithm is a monotonic function, maximising the likelihood is equivalent to maximising the log-likelihood.
Thus the optimisation problem becomes

$$\theta^* = \arg\max_{\theta} \sum_{i=1}^{n} \log p(y_i|x_i, \theta).$$

This additive form is much easier to optimise using gradient-based algorithms.

Deep Dive

Another advantage of log probabilities is numerical stability.
Probabilities are always less than or equal to 1. When many probabilities are multiplied together, the result can quickly become extremely close to zero, leading to numerical underflow.
Logarithms transform these tiny numbers into manageable values and prevent underflow during optimisation.

3.0.66 Worked Example

Worked Example

Suppose three independent observations have probabilities

$$0.8, \quad 0.7, \quad 0.6.$$

The likelihood is

$$0.8 \times 0.7 \times 0.6 = 0.336.$$

The log-likelihood is

$$\log(0.8) + \log(0.7) + \log(0.6).$$

Instead of multiplying small numbers together, optimisation algorithms can sum log probabilities, which is computationally more stable.

3.0.67 Python Demonstration

```python
import numpy as np

probs = np.array([0.8, 0.7, 0.6])

likelihood = np.prod(probs)
log_likelihood = np.sum(np.log(probs))

print("Likelihood:", likelihood)
print("Log-likelihood:", log_likelihood)
```

Listing 3.8: Likelihood vs log-likelihood

3.0.68 Follow-Up Interview Questions

Follow-Up Interview Questions

- Why is log-likelihood used instead of likelihood in optimisation?
- What is numerical underflow and why does it occur?
- Why does maximising likelihood produce the same solution as maximising log-likelihood?
- How does log-loss arise from the log-likelihood of a Bernoulli distribution?

3.0.69 Common Mistakes

Common Mistake

Thinking that taking the logarithm changes the optimal solution. Because the logarithm is monotonic, the location of the optimum remains the same.

Common Mistake

Ignoring the numerical stability benefits of log probabilities when working with large datasets.

3.0.70 Summary

Log probabilities are widely used in machine learning optimisation because they simplify likelihood calculations and improve numerical stability. Key points to remember in a technical interview include:

- Logarithms convert products of probabilities into sums.
- Maximising likelihood is equivalent to maximising log-likelihood.
- Log probabilities prevent numerical underflow.
- Many machine learning loss functions arise from negative log-likelihood.

Interview Question

Question 12: What is log-loss and how is it related to cross-entropy?

Difficulty Level

Tier: Core Question

3.0.71 Short Interview Answer

Short Interview Answer

Log-loss is a loss function used in classification that measures how well predicted probabilities match the true class labels. It penalises incorrect predictions based on the negative logarithm of the predicted probability assigned to the correct class.
Log-loss is mathematically equivalent to cross-entropy when the true labels are represented as one-hot encoded probability distributions.

3.0.72 Intuition

In classification problems, models typically output probabilities for each class. For example, a binary classifier might produce the prediction

$$P(y = 1 \mid x) = 0.8.$$

If the true label is $y = 1$, this prediction is good because the model assigned high probability to the correct class.

If the model instead predicts

$$P(y = 1 \mid x) = 0.1,$$

then the prediction is very poor because the model was confident in the wrong answer.

Log-loss measures this error by taking the negative logarithm of the probability assigned to the correct class:

$$L = -\log(\hat{p}_{\text{correct}}).$$

This creates two desirable properties:

- Correct predictions with high probability produce very small loss.
- Incorrect predictions with low probability produce very large loss.

Interview Tip

In interviews, a strong answer notes that log-loss is simply the practical form of cross-entropy used when training classifiers.

3.0.73 Mathematical Perspective

Mathematical Insight

For binary classification, log-loss is defined as

$$L(y, \hat{p}) = -\big[y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})\big],$$

where

- y is the true label (0 or 1)
- $\hat{p}$ is the predicted probability for the positive class

For multi-class classification, log-loss becomes

$$L = -\sum_{i} y_i \log(\hat{p}_i),$$

where y_i is a one-hot encoded label.
This expression is exactly the same as the cross-entropy between the true distribution p and the predicted distribution q:

$$H(p, q) = -\sum_{i} p_i \log q_i.$$

Thus log-loss and cross-entropy are mathematically equivalent when the true labels are represented as probability distributions.

Deep Dive

Log-loss can also be interpreted from a probabilistic perspective.
If a classifier outputs predicted probabilities $\hat{p}$, we can treat these probabilities as parameters of a Bernoulli or categorical distribution.
Maximising the likelihood of the observed labels leads directly to the log-loss objective. Therefore minimising log-loss is equivalent to performing maximum likelihood estimation for probabilistic classifiers.

3.0.74 Worked Example

Worked Example

Suppose a classifier predicts the probability of the positive class.

True Label	Predicted Probability	Log-Loss
1	0.9	$-\log(0.9) = 0.105$
1	0.6	$-\log(0.6) = 0.511$
1	0.1	$-\log(0.1) = 2.303$

The loss increases rapidly as the predicted probability for the correct class becomes smaller.

3.0.75 Python Demonstration

```python
import numpy as np

# true labels
y = np.array([1, 0, 1])

# predicted probabilities
p = np.array([0.9, 0.2, 0.6])

loss = -(y*np.log(p) + (1-y)*np.log(1-p))

print("Log-loss values:", loss)
```

Listing 3.9: Computing log-loss

3.0.76 Follow-Up Interview Questions

Follow-Up Interview Questions

- Why does log-loss penalise confident incorrect predictions so strongly?
- How does log-loss relate to maximum likelihood estimation?
- Why is log-loss preferred over squared error for classification?
- How does log-loss behave when predicted probabilities approach zero?

3.0.77 Common Mistakes

Common Mistake

Treating log-loss and cross-entropy as completely different loss functions when they are mathematically equivalent in classification settings.

Common Mistake

Failing to explain that log-loss evaluates predicted probabilities rather than raw class predictions.

3.0.78 Summary

Log-loss is a classification loss function that penalises incorrect probabilistic predictions. Key points to remember include:

- Log-loss measures the negative log probability assigned to the correct class.
- It heavily penalises confident incorrect predictions.
- Log-loss is mathematically equivalent to cross-entropy when labels are one-hot encoded.
- Minimising log-loss corresponds to maximum likelihood estimation for probabilistic classifiers.

Chapter 4

Gradient Descent

Introduction

Training a machine learning model typically involves solving an optimisation problem. Earlier in this Part in Chapter 3 we introduced loss functions such as mean squared error and cross-entropy, which measure how well a model's predictions match the observed data. The goal of training is therefore to find model parameters that minimise the chosen loss function.

For simple models, this optimisation problem may have a closed-form solution. However, for many machine learning models — particularly neural networks — no analytic solution exists. Instead, the parameters must be found using iterative optimisation algorithms. The most widely used optimisation algorithm in machine learning is **gradient descent**.

Gradient descent works by repeatedly adjusting the model parameters in the direction that most rapidly decreases the loss. By following the gradient of the loss function, the algorithm gradually moves toward a set of parameters that minimise prediction error.

Understanding gradient descent is essential for understanding how modern machine learning models are trained. It also appears frequently in technical interviews for data science and machine learning roles. In this chapter we explore:

- how gradient descent works
- why gradients indicate the direction of steepest descent
- different variants such as stochastic and mini-batch gradient descent
- common optimisation challenges such as learning rates and local minima

Interview Question

Question 13: What is gradient descent?

Difficulty Level

Tier: Core Question

4.0.1 Short Interview Answer

Short Interview Answer

Gradient descent is an optimisation algorithm used to minimise a loss function by iteratively updating model parameters in the direction of the negative gradient.
At each step, the algorithm computes the gradient of the loss with respect to the parameters and moves the parameters slightly in the direction that reduces the loss.

4.0.2 Intuition

Gradient descent can be understood using the analogy of descending a hill. Imagine standing somewhere on a mountainous landscape and wanting to reach the lowest point in the valley. If you look at the slope of the terrain around you, the steepest downward direction indicates the fastest way to decrease your elevation.

The gradient of a function provides exactly this information: it tells us the direction of steepest increase. By moving in the opposite direction, we move toward lower values of the function.

In machine learning, the function we want to minimise is the **loss function**. Gradient descent repeatedly updates the model parameters in the direction that reduces this loss.

Interview Tip

In interviews, it is helpful to begin with the intuition of moving downhill on a loss surface before presenting the mathematical update rule.

4.0.3 Mathematical Perspective

Mathematical Insight

Suppose a model has parameters θ and a loss function $J(\theta)$.
Gradient descent updates the parameters according to the rule

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t),$$

where

- $\nabla J(\theta_t)$ is the gradient of the loss function
- η is the learning rate

The gradient indicates the direction of steepest increase of the loss. Moving in the opposite direction therefore reduces the loss.

By repeating this update many times, the algorithm gradually moves toward a set of parameters that minimise the loss function.

Deep Dive

For convex loss functions, gradient descent is guaranteed to converge to the global minimum. For non-convex functions — such as those encountered in deep learning — gradient descent may instead converge to a local minimum or saddle point. Despite this, it remains highly effective in practice.

4.0.4 Worked Example

Worked Example

Consider the simple function

$$J(\theta) = (\theta - 3)^2.$$

The gradient of this function is

$$\frac{dJ}{d\theta} = 2(\theta - 3).$$

Starting from an initial value $\theta_0 = 0$, gradient descent repeatedly updates the parameter in the direction that reduces the function value until it approaches the minimum at $\theta = 3$.

4.0.5 Python Demonstration

```
import numpy as np

# learning rate
eta = 0.1

# initial parameter
theta = 0

for i in range(20):

    gradient = 2*(theta - 3)
    theta = theta - eta*gradient
```

```
13
14 print("Estimated minimum:", theta)
```

Listing 4.1: Simple gradient descent example

4.0.6 Visual Interpretation

Gradient descent can be visualised as a sequence of steps moving downhill along the loss surface. Each step moves the parameters slightly toward regions where the loss function is smaller. As the algorithm approaches the minimum, the gradients become smaller and the updates gradually shrink. This process continues until the parameters converge to a minimum of the loss function.

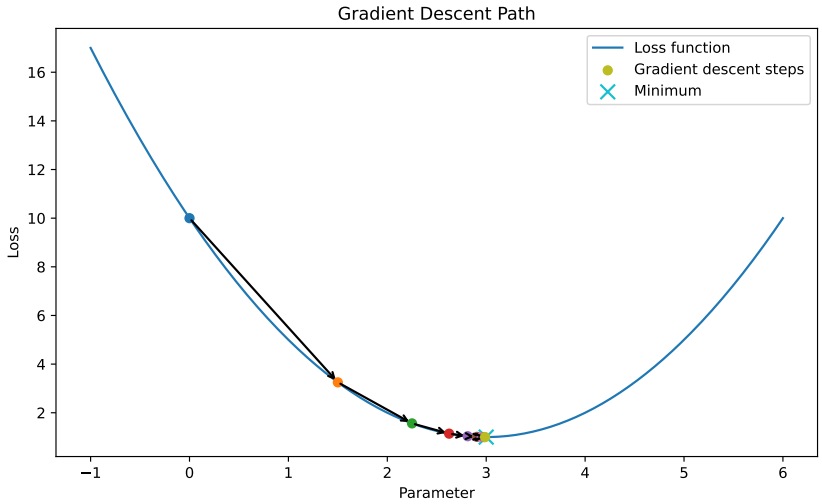


Figure 4.1: Gradient descent iteratively moves in the direction of the negative gradient to reach the minimum of the loss function.

4.0.7 Follow-Up Interview Questions

Follow-Up Interview Questions

- Why does the gradient indicate the direction of steepest increase?
- What role does the learning rate play in gradient descent?
- What happens if the learning rate is too large?
- What is the difference between batch, stochastic, and mini-batch gradient descent?

4.0.8 Common Mistakes

Common Mistake

Describing gradient descent only as an algorithm for linear regression rather than a general optimisation method used across machine learning.

Common Mistake

Failing to explain why the negative gradient direction reduces the loss.

4.0.9 Summary

Gradient descent is a general optimisation algorithm used to minimise loss functions in machine learning. The key ideas to remember for a technical interview include:

- The gradient indicates the direction of steepest increase of the loss.
- Moving in the opposite direction reduces the loss.
- Gradient descent repeatedly updates parameters to minimise the loss function.
- The learning rate controls the size of each update step.

Interview Question

Question 14: Why does the gradient point in the direction of steepest ascent?

Difficulty Level

Tier: Advanced

4.0.10 Short Interview Answer

Short Interview Answer

The gradient of a function points in the direction of steepest ascent because it contains the partial derivatives of the function with respect to each parameter. These derivatives measure how rapidly the function changes in each direction.
The direction of the gradient therefore corresponds to the direction in which the function increases most rapidly. Moving in the opposite direction of the gradient leads to the steepest decrease in the function, which is why gradient descent follows the negative gradient.

4.0.11 Intuition

To understand why the gradient indicates the steepest ascent, consider a function of two variables

$$f(x, y).$$

At any point on the surface defined by this function, there are infinitely many directions in which we could move. Each direction will cause the function value to change at a different rate. The gradient vector collects the partial derivatives

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right).$$

These derivatives measure how sensitive the function is to changes in each variable. When combined into a vector, they indicate the direction in which the function increases most rapidly. An intuitive way to visualise this is to imagine standing on a hill. The gradient points in the direction of the steepest uphill slope.

Interview Tip

In interviews, it is useful to explain that the gradient tells us how the function changes in each coordinate direction, and that combining these changes produces the direction of maximum increase.

4.0.12 Mathematical Perspective

Mathematical Insight

Let $f(\mathbf{x})$ be a differentiable function and let $\mathbf{v}$ be a unit vector representing a direction. The rate of change of the function in that direction is given by the directional derivative

$$D_{\mathbf{v}} f = \nabla f \cdot \mathbf{v}.$$

Using the Cauchy–Schwarz inequality,

$$\nabla f \cdot \mathbf{v} \leq \|\nabla f\|.$$

Equality occurs when

$$\mathbf{v} = \frac{\nabla f}{\|\nabla f\|}.$$

This shows that the directional derivative is maximised when the direction vector aligns with the gradient.
Therefore the gradient points in the direction of steepest ascent.

Because the gradient points in the direction of fastest increase, moving in the opposite direction

$$-\nabla f$$

produces the fastest decrease in the function.

Deep Dive

This property explains why gradient descent uses the update rule

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t).$$

The gradient indicates the direction of maximum increase of the loss function J. Subtracting the gradient therefore moves the parameters toward regions where the loss is smaller.

4.0.13 Worked Example

Worked Example

Consider the function

$$f(x, y) = x^2 + y^2.$$

The gradient is

$$\nabla f = (2x, 2y).$$

At the point $(1, 1)$, the gradient becomes

$$\nabla f = (2, 2).$$

This vector points directly away from the origin, which is the direction in which the function increases most rapidly.
Moving in the opposite direction $(-2, -2)$ leads toward the minimum at the origin.

4.0.14 Follow-Up Interview Questions

Follow-Up Interview Questions

- What is a directional derivative?
- Why is the gradient perpendicular to level curves?
- Why does gradient descent follow the negative gradient?
- How does this idea extend to high-dimensional parameter spaces?

4.0.15 Common Mistakes

Common Mistake

Saying that the gradient simply indicates the slope without explaining why it corresponds to the direction of steepest ascent.

Common Mistake

Confusing the gradient direction with the path taken by gradient descent, which depends on the learning rate and optimisation dynamics.

4.0.16 Summary

The gradient of a function indicates the direction of steepest ascent because it collects the partial derivatives with respect to each parameter into a single vector that describes how the function changes locally. More precisely, the directional derivative is maximised when we move in the direction aligned with the gradient, which is why the gradient points toward the direction of maximum increase. This geometric interpretation is fundamental in optimisation: if we want to minimise a loss function rather than increase it, gradient descent moves in the opposite direction of the gradient. In an interview, the key idea to remember is that the gradient is not just a collection of derivatives, but the local direction in parameter space that most rapidly increases the function.

Interview Question

Question 15: What is the role of the learning rate in gradient descent?

Difficulty Level

Tier: Core Question

4.0.17 Short Interview Answer

Short Interview Answer

The learning rate controls the size of the parameter updates during gradient descent. It determines how far the algorithm moves in the direction of the negative gradient at each iteration.

If the learning rate is too small, training becomes very slow. If it is too large, the algorithm may overshoot the minimum or even diverge.

4.0.18 Intuition

When performing gradient descent, we repeatedly update the parameters of a model to reduce the loss. However, we must decide how large each update step should be. The learning rate determines this step size. Imagine walking downhill toward the bottom of a valley.

- If you take extremely small steps, you will eventually reach the bottom, but it will take a very long time.
- If you take very large steps, you may overshoot the bottom of the valley and bounce back and forth without ever settling.

The learning rate therefore controls how quickly the optimisation algorithm moves toward the minimum of the loss function.

Interview Tip

In interviews, it is helpful to mention both failure modes: very small learning rates lead to slow convergence, while very large learning rates can cause divergence.

4.0.19 Mathematical Perspective

Mathematical Insight

Recall the gradient descent update rule

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t),$$

where

- θ_t represents the current parameters
- $\nabla J(\theta_t)$ is the gradient of the loss
- η is the learning rate

The learning rate scales the gradient vector and therefore determines the magnitude of the parameter update.

Choosing an appropriate learning rate is therefore critical for effective optimisation.

Deep Dive

In practice, many optimisation algorithms use adaptive learning rates that adjust automatically during training.
Examples include:

- AdaGrad
- RMSProp
- Adam

These methods modify the effective learning rate based on the history of gradients, often improving convergence in large-scale machine learning problems.

4.0.20 Worked Example

Worked Example

Consider the function

$$J(\theta) = (\theta - 5)^2.$$

Suppose the current parameter value is

$$\theta = 0.$$

The gradient is

$$\nabla J(\theta) = 2(\theta - 5) = -10.$$

If the learning rate is

$$\eta = 0.1,$$

the update becomes

$$\theta_1 = 0 - 0.1(-10) = 1.$$

A larger learning rate would produce a larger update step.

4.0.21 Python Demonstration

```python
import numpy as np

def gradient(theta):
    return 2*(theta - 5)

theta = 0
eta = 0.1

for i in range(10):
    theta = theta - eta * gradient(theta)

print("Estimated minimum:", theta)
```

Listing 4.2: Effect of learning rate on gradient descent

4.0.22 Visual Interpretation

The learning rate controls the size of the steps taken along the loss surface.

- Small learning rate → slow but stable convergence
- Large learning rate → faster progress but risk of overshooting

Choosing an appropriate learning rate is therefore one of the most important hyperparameters in gradient-based optimisation.

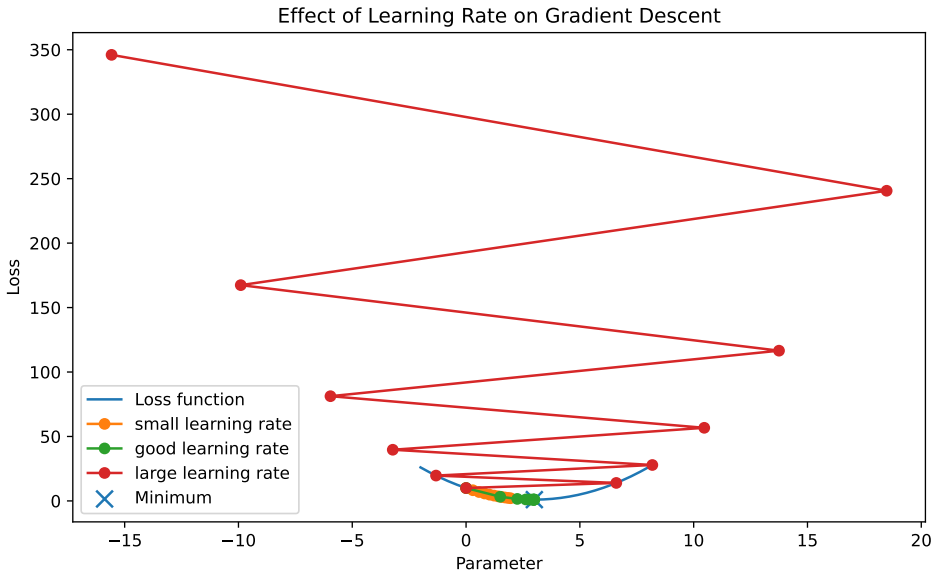


Figure 4.2: Effect of learning rate on gradient descent. Small learning rates converge slowly, while very large learning rates can overshoot the minimum.

4.0.23 Follow-Up Interview Questions

Follow-Up Interview Questions

- What happens if the learning rate is too large?
- What happens if the learning rate is too small?
- What is learning rate scheduling?
- How do adaptive optimisation algorithms adjust the learning rate?

4.0.24 Common Mistakes

Common Mistake

Assuming that a larger learning rate always leads to faster training. In practice, large learning rates can cause the optimisation to diverge.

Common Mistake

Ignoring the importance of learning rate tuning when training machine learning models.

4.0.25 Summary

The learning rate controls how large each gradient descent update step is. Key ideas to remember include:

- The learning rate scales the gradient during parameter updates.
- Small learning rates lead to slow convergence.
- Large learning rates can cause overshooting or divergence.
- Choosing an appropriate learning rate is critical for efficient optimisation.

Interview Question

Question 16: What is stochastic gradient descent?

Difficulty Level

Tier: Core Question

4.0.26 Short Interview Answer

Short Interview Answer

Stochastic gradient descent (SGD) is a variant of gradient descent in which the model parameters are updated using the gradient computed from a single training example rather than the entire dataset.
This makes each update much faster and allows the algorithm to scale to large datasets, although the updates become noisier.

4.0.27 Intuition

In standard gradient descent, the gradient of the loss function is computed using the entire training dataset before updating the parameters. If the dataset is very large, this can be computationally expensive because every update requires processing all training examples.

Stochastic gradient descent solves this problem by updating the parameters using only one training example at a time. Instead of computing

$$\nabla J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla L_i(\theta),$$

SGD updates the parameters using the gradient from a single observation:

$$\nabla L_i(\theta).$$

Because each update uses only one example, the updates become much faster. However, they also become noisier because each example provides only a rough estimate of the true gradient.

Interview Tip

In interviews, emphasise that SGD uses an unbiased but noisy estimate of the true gradient.

4.0.28 Mathematical Perspective

Mathematical Insight

Suppose the loss for a single training example is

$$L_i(\theta).$$

Stochastic gradient descent updates the parameters according to

$$\theta_{t+1} = \theta_t - \eta \nabla L_i(\theta_t),$$

where i is a randomly selected training example.
Because the expectation of this gradient equals the true gradient,

$$\mathbb{E}[\nabla L_i(\theta)] = \nabla J(\theta),$$

SGD provides an unbiased estimate of the full gradient.

Although the updates are noisy, they still move the parameters in the correct direction on average.

Deep Dive

The randomness in stochastic gradient descent can sometimes be beneficial.
The noise in the updates can help the optimisation algorithm escape shallow local minima or saddle points, which can improve optimisation in complex loss landscapes such as those encountered in deep learning.

4.0.29 Worked Example

Worked Example

Suppose a dataset contains three training examples with losses

$$L_1(\theta), \quad L_2(\theta), \quad L_3(\theta).$$

Batch gradient descent would compute the gradient

$$\nabla J(\theta) = \frac{1}{3} \left(\nabla L_1 + \nabla L_2 + \nabla L_3 \right).$$

Stochastic gradient descent instead updates the parameters using only one of these gradients at each step.
For example:

- Step 1: use ∇L_1
- Step 2: use ∇L_3

- Step 3: use ∇L_2

The updates therefore fluctuate around the true gradient direction.

4.0.30 Python Demonstration

This code demonstrates a very simple form of **stochastic gradient descent** for fitting a one-parameter linear model. The dataset follows the exact relationship $y = 2x$, and the model assumes predictions of the form

$$\hat{y} = \theta x.$$

The parameter θ is initialised at 0, and the code then repeatedly updates it by looping through the training examples one at a time. For each example, it computes the current prediction, calculates the gradient of the squared error with respect to θ, and then moves θ a small step in the direction that reduces the error. Because the updates are done after each individual training example rather than after the whole dataset, this is stochastic gradient descent rather than full batch gradient descent. Over the ten epochs, θ moves closer to the true value 2, so the final printed result is an estimate of the slope of the underlying linear relationship.

```
import numpy as np

# simple dataset
X = np.array([1, 2, 3, 4])
y = np.array([2, 4, 6, 8])

theta = 0
eta = 0.01

for epoch in range(10):

    for i in range(len(X)):

        prediction = theta * X[i]
        gradient = 2 * X[i] * (prediction - y[i])

        theta = theta - eta * gradient

print("Estimated parameter:", theta)
```

Listing 4.3: Simple stochastic gradient descent example

4.0.31 Follow-Up Interview Questions

Follow-Up Interview Questions

- Why does stochastic gradient descent introduce noise into the optimisation process?
- What is the difference between stochastic and mini-batch gradient descent?
- Why is SGD commonly used for training neural networks?
- How does dataset size influence the choice of optimisation algorithm?

4.0.32 Common Mistakes

Common Mistake

Thinking that stochastic gradient descent always converges faster than batch gradient descent. While each update is cheaper, the noisy updates may require more iterations.

Common Mistake

Confusing stochastic gradient descent with mini-batch gradient descent, which uses small batches rather than single examples.

4.0.33 Summary

Stochastic gradient descent is an optimisation method that updates model parameters using gradients computed from individual training examples. Key ideas to remember for a technical interview include:

- SGD updates parameters using one training example at a time.
- This greatly reduces the cost of each update.
- The updates become noisy but remain unbiased estimates of the true gradient.
- SGD is widely used for training large-scale machine learning models.

Interview Question

Question 17: What is the difference between batch, stochastic, and mini-batch gradient descent?

Difficulty Level

Tier: Core Question

4.0.34 Short Interview Answer

Short Interview Answer

Batch gradient descent updates model parameters using the gradient computed from the entire training dataset. Stochastic gradient descent updates parameters using the gradient from a single training example. Mini-batch gradient descent uses a small subset of training examples for each update.

Mini-batch gradient descent is the most commonly used approach in practice because it provides a good balance between computational efficiency and stable gradient estimates.

4.0.35 Intuition

Gradient descent requires computing gradients of the loss function with respect to the model parameters. The main difference between the three variants lies in how much data is used to compute each gradient update.

- **Batch gradient descent** computes the gradient using the entire dataset. This produces a precise estimate of the true gradient but can be slow for large datasets.
- **Stochastic gradient descent** computes the gradient using a single training example. Updates are extremely fast but noisy because each example provides only a rough estimate of the gradient.
- **Mini-batch gradient descent** computes the gradient using a small subset of the data, often between 32 and 512 examples. This reduces noise while still allowing efficient computation.

Mini-batch gradient descent is widely used in modern machine learning because it allows efficient training on large datasets while maintaining relatively stable updates.

Interview Tip

In interviews, it is useful to emphasise that mini-batch gradient descent is the standard approach used in deep learning frameworks such as PyTorch and TensorFlow.

4.0.36 Mathematical Perspective

Mathematical Insight

Suppose the training dataset contains n examples with individual losses $L_i(\theta)$.
The full objective is

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L_i(\theta).$$

The gradient update differs depending on the optimisation method.
Batch gradient descent

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t).$$

Stochastic gradient descent

$$\theta_{t+1} = \theta_t - \eta \nabla L_i(\theta_t).$$

Mini-batch gradient descent

$$\theta_{t+1} = \theta_t - \eta \frac{1}{|B|}\sum_{i \in B} \nabla L_i(\theta_t),$$

where B is a small batch of training examples.

4.0.37 Worked Example

Worked Example

Suppose a dataset contains 10,000 training examples.

- Batch gradient descent computes the gradient using all 10,000 examples before every update.
- Stochastic gradient descent computes the gradient using only one example at a time.
- Mini-batch gradient descent might compute the gradient using a batch of 64 examples.

Thus mini-batch gradient descent performs many more updates than batch gradient descent while maintaining more stable gradients than SGD.

4.0.38 Python Demonstration

```
import numpy as np

X = np.random.randn(1000)
y = 2*X + np.random.randn(1000)*0.1

```

```
theta = 0
eta = 0.01
batch_size = 32

for epoch in range(10):

    for i in range(0, len(X), batch_size):

        X_batch = X[i:i+batch_size]
        y_batch = y[i:i+batch_size]

        pred = theta * X_batch
        gradient = np.mean(2*X_batch*(pred - y_batch))

        theta = theta - eta*gradient

print("Estimated parameter:", theta)
```

Listing 4.4: Illustrating mini-batch gradient descent

4.0.39 Visual Interpretation

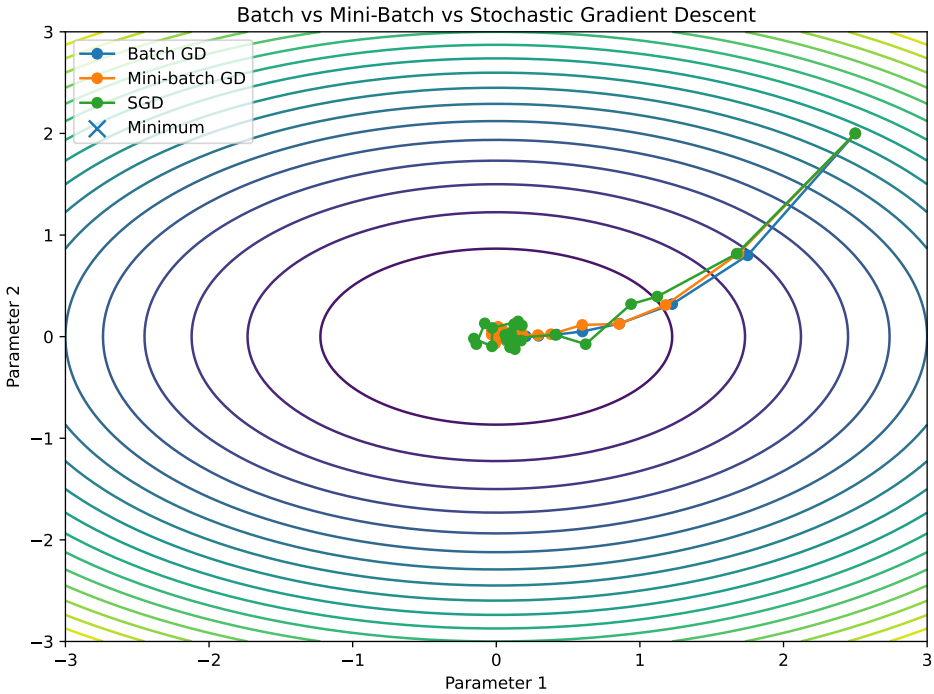


Figure 4.3: Comparison of optimisation paths for batch, stochastic, and mini-batch gradient descent. Mini-batch methods balance stability and computational efficiency.

The optimisation trajectories of these algorithms differ:

- Batch gradient descent follows a smooth path toward the minimum.

- Stochastic gradient descent produces noisy, fluctuating updates.
- Mini-batch gradient descent produces moderately smooth updates while remaining computationally efficient.

This balance explains why mini-batch methods dominate modern machine learning optimisation.

4.0.40 Follow-Up Interview Questions

Follow-Up Interview Questions

- Why is mini-batch gradient descent typically preferred in practice?
- How does batch size affect training stability?
- Why do GPUs perform well with mini-batch training?
- What happens when the batch size becomes very large?

4.0.41 Common Mistakes

Common Mistake

Assuming that stochastic gradient descent is always faster than mini-batch methods. In practice, modern hardware is optimised for vectorised operations on batches.

Common Mistake

Ignoring the trade-off between computational cost and gradient variance when choosing batch size.

4.0.42 Summary

Batch, stochastic, and mini-batch gradient descent differ in how much data is used to compute each gradient update. The key ideas to remember include:

- Batch gradient descent uses the entire dataset.
- Stochastic gradient descent uses a single example.
- Mini-batch gradient descent uses small subsets of the data.
- Mini-batch methods are the most common approach in modern machine learning.

Part III

Evaluation and Metrics

Once a model has been specified and trained, the next question is how to judge whether it is actually any good. This sounds straightforward, but in machine learning evaluation is rarely captured by a single number. Different metrics highlight different aspects of performance, and the most appropriate choice depends on the structure of the problem, the data distribution, and the practical cost of different kinds of errors. For that reason, evaluation is not just the final stage of modelling; it is a central part of how we define success in the first place.

This part of the book focuses on the tools used to measure, compare, and interpret model performance. It covers the core regression and classification metrics that appear frequently in interviews, but it also goes further by examining what those metrics really mean, when they become misleading, and how they should be used in practice. Accuracy, precision, recall, F1 score, ROC curves, precision–recall curves, calibration, significance testing, and model comparison all belong to this broader question of evaluation. A strong interview answer in this area does more than state a formula: it explains what the metric captures, what it ignores, and why it matters in a real decision-making context.

Evaluation is also where the connection between statistics and machine learning becomes especially visible. Questions about cross-validation, confidence, significance, calibration, and uncertainty all concern whether an observed result can be trusted. A model may perform well on one split and poorly on another, may look strong under one metric and weak under another, or may produce impressive scores while still failing in the ways that matter most operationally. This is why evaluation requires judgement as well as calculation.

In interviews, this part is particularly important because it tests whether a candidate can think beyond raw optimisation and speak clearly about evidence. It is one thing to train a model; it is another to justify why it should be trusted, how it should be compared with alternatives, and whether its reported performance is genuinely meaningful. The aim of Part III is therefore to build a deeper understanding of model assessment, so that evaluation becomes not a box-ticking exercise, but a principled way of reasoning about model quality.

Chapter 5

Information Criteria

Introduction

In model selection, a recurring challenge is deciding whether a more complex model is genuinely better or simply fitting noise in the observed data. Training likelihood almost always improves as we add parameters, but this does not necessarily mean the model will generalise well or provide the most useful explanation of the data.

Information criteria such as **AIC** and **BIC** were developed to address this trade-off. They combine a measure of model fit with a penalty for complexity, allowing us to compare models in a more principled way than by goodness of fit alone. These criteria are especially important in regression, probabilistic modelling, and time series analysis, where likelihood-based model comparison is common. In interviews, they are often used to test whether a candidate understands the distinction between fitting the observed data well and choosing a model that is appropriately parsimonious.

Interview Question

Question 18: What are AIC and BIC?

Difficulty Level

Tier: Core Question

5.0.1 Short Interview Answer

Short Interview Answer

AIC and BIC are information criteria used for model selection. Both combine a measure of goodness of fit, usually based on the log-likelihood, with a penalty for model complexity. The goal is to prefer models that explain the data well without using unnecessary parameters. Lower AIC or BIC values indicate a better trade-off between fit and complexity among the models being compared.

5.0.2 Intuition

AIC and BIC are designed to answer a very practical question: when two models fit the data reasonably well, how should we decide whether the more complicated one is actually worth using? If we judge models only by how well they fit the observed data, then more flexible models will often look better simply because they have more freedom to adapt to noise. Information criteria correct for this by rewarding good fit but penalising unnecessary complexity.

This makes them different from prediction metrics such as MSE or accuracy. Those metrics tell us how well a model performs on a given task, whereas AIC and BIC are tools for comparing competing probabilistic models fitted to the same dataset. Their purpose is not to measure error directly, but to formalise the trade-off between explanatory power and parsimony.

Interview Tip

A strong interview answer should emphasise that AIC and BIC are **model selection criteria**, not generic performance metrics. They reward good likelihood but penalise extra parameters.

5.0.3 Mathematical Perspective

Mathematical Insight

Suppose a fitted model has maximised likelihood L, and let k denote the number of estimated parameters. Then the two most common information criteria are:

$$\text{AIC} = 2k - 2\log L$$

and

$$\text{BIC} = k\log n - 2\log L,$$

where n is the number of observations.

In both expressions, the term $-2\log L$ measures lack of fit, so a larger likelihood leads to a smaller value and is therefore preferred. The remaining term penalises model complexity. AIC uses a penalty of $2k$, while BIC uses $k\log n$, which typically grows more strongly with sample size.

The key idea is that information criteria do not ask only, "How well does the model fit?" They ask, "How well does the model fit relative to how complex it is?"

5.0.4 Worked Example

Worked Example

Suppose we compare two models fitted to the same dataset.

Model A has:

$$k = 3, \qquad \log L = -120$$

Model B has:

$$k = 6, \qquad \log L = -115$$

For Model A:

$$\text{AIC}_A = 2(3) - 2(-120) = 6 + 240 = 246$$

For Model B:

$$\text{AIC}_B = 2(6) - 2(-115) = 12 + 230 = 242$$

So under AIC, Model B is preferred because it improves the fit enough to justify the extra parameters.

Now suppose the dataset has $n = 50$, so that $\log n \approx 3.91$.

Then:

$$\text{BIC}_A = 3\log 50 - 2(-120) \approx 3(3.91) + 240 = 251.73$$

$$\text{BIC}_B = 6\log 50 - 2(-115) \approx 6(3.91) + 230 = 253.46$$

Under BIC, Model A is preferred. This illustrates an important practical point: AIC and BIC can disagree because BIC penalises complexity more strongly, especially as the sample size grows.

5.0.5 Python Demonstration

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Generate synthetic non-linear data
rng = np.random.default_rng(0)
X = np.linspace(-3, 3, 80)
y_true = 1.5 + 2.0 * X - 0.8 * X**2
y = y_true + rng.normal(0, 2.0, size=len(X))

n = len(X)
degrees = range(1, 7)
aic_values = []
bic_values = []

for degree in degrees:
    poly = PolynomialFeatures(degree=degree, include_bias=True)
    X_poly = poly.fit_transform(X.reshape(-1, 1))

    model = LinearRegression(fit_intercept=False)
    model.fit(X_poly, y)
    y_pred = model.predict(X_poly)

    residuals = y - y_pred
    rss = np.sum(residuals**2)

    # Gaussian-noise log-likelihood with estimated variance
    sigma2 = rss / n
    logL = -n / 2 * (np.log(2 * np.pi * sigma2) + 1)

    k = X_poly.shape[1]

    aic = 2 * k - 2 * logL
    bic = k * np.log(n) - 2 * logL

    aic_values.append(aic)
    bic_values.append(bic)


plt.show()
```

Listing 5.1: Comparing AIC and BIC for polynomial regression models

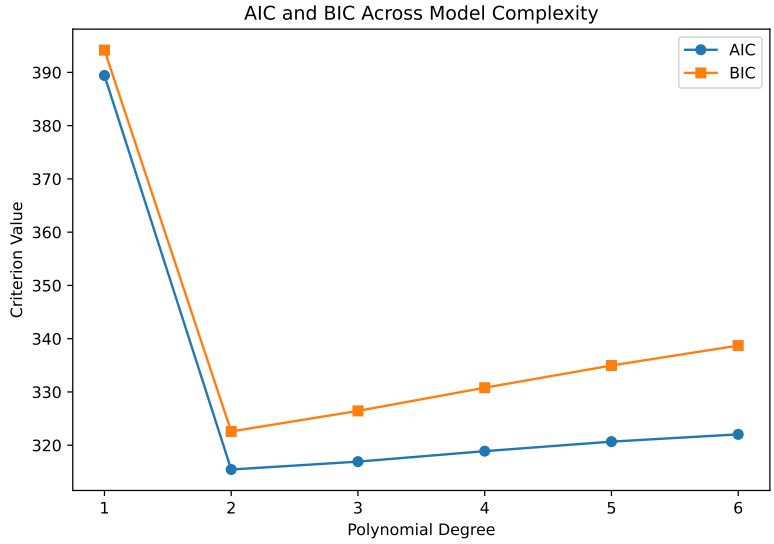


Figure 5.1: AIC and BIC values for polynomial regression models of increasing degree. Both criteria reward improved fit but penalise unnecessary complexity, so lower values indicate a better balance between the two.

This example shows the central purpose of information criteria: as model complexity increases, fit often improves at first, but after a certain point the penalty for additional parameters begins to outweigh the gain. The preferred model is therefore not necessarily the simplest one or the one with the highest likelihood, but the one that achieves the best trade-off.

5.0.6 Deep Dive: Why Are They Called Information Criteria?

Deep Dive

The term *information criterion* comes from information-theoretic and statistical arguments about how much information is lost when we use a candidate model to represent the data-generating process.

AIC is motivated by the idea of estimating predictive information loss. More precisely, it arises from an approximation to the expected Kullback–Leibler divergence between the true data-generating distribution and the fitted model. In that sense, AIC tries to select the model that is expected to be closest to the truth in predictive terms, even if none of the candidate models is exactly correct.

BIC has a different motivation. It comes from a Bayesian approximation to the marginal likelihood of a model. While it is often used in a similar practical way, its interpretation is not identical to that of AIC. In interviews, it is usually enough to say that both are likelihood-based model selection criteria, but that AIC has a stronger predictive-information interpretation whereas BIC has a stronger Bayesian model-selection flavour.

The deeper lesson is that both criteria are trying to correct the same fundamental problem:

raw training fit is too optimistic when model complexity is ignored.

5.0.7 Follow-Up Interview Questions

Follow-Up Interview Questions

- Why do lower AIC and BIC values indicate a better model?
- Why can AIC and BIC disagree when comparing the same models?
- What does the penalty term represent?
- Are AIC and BIC prediction metrics like MSE or accuracy?

5.0.8 Common Mistakes

Common Mistake

Treating AIC or BIC as absolute measures of model quality rather than relative criteria for comparing models fitted to the same data.

Common Mistake

Forgetting that lower values are better.

Common Mistake

Saying that AIC or BIC directly measure prediction error.

Common Mistake

Comparing AIC or BIC across models fitted to different datasets.

5.0.9 Summary

AIC and BIC are information criteria used for model selection, not direct performance metrics. Both combine a likelihood-based measure of fit with a penalty for complexity, so that models are rewarded for explaining the data well but discouraged from using unnecessary parameters. This makes them useful when comparing competing probabilistic models fitted to the same dataset. In practice, they provide a principled way to balance fit and parsimony, with lower values indicating a better trade-off.

Interview Question

Question 19: How do information criteria compare with cross-validation for model selection?

Difficulty Level

Tier: Advanced

5.0.10 Short Interview Answer

Short Interview Answer

Information criteria such as AIC and BIC and cross-validation are both used for model selection, but they approach the problem differently. Information criteria compare models using likelihood together with an explicit penalty for complexity, while cross-validation estimates how well a model is likely to perform on unseen data by repeatedly evaluating it on held-out subsets. In general, information criteria are more analytic and computationally efficient, whereas cross-validation is more directly tied to predictive performance. The better choice depends on whether the goal is statistical model selection, predictive accuracy, or a balance between the two.

5.0.11 Intuition

Information criteria and cross-validation are both designed to answer the same broad question: which model should we prefer? The difference is that they define "better" in different ways. Information criteria start from the fitted likelihood and then penalise complexity, so they reward models that explain the observed data well without becoming unnecessarily flexible. Cross-validation takes a more empirical approach. Instead of using a formula that adjusts training fit, it repeatedly withholds part of the data, trains the model on the remainder, and measures performance on the held-out portion.

This leads to an important conceptual distinction. Information criteria are usually most natural when working with probabilistic statistical models, where likelihood is central and the number of parameters is meaningful. Cross-validation is more general and more directly aligned with predictive modelling, because it asks how well the model performs on data it did not see during training. In that sense, information criteria are often about balancing fit and parsimony within a modelling framework, while cross-validation is often about estimating real predictive usefulness.

Interview Tip

A strong interview answer should say that information criteria are **likelihood-based penalised fit measures**, whereas cross-validation is a **held-out performance estimate**. Information criteria are often more efficient; cross-validation is often more directly predictive.

5.0.12 Mathematical Perspective

Mathematical Insight

Information criteria typically take the form

$$\text{criterion} = \text{lack of fit} + \text{complexity penalty}.$$

For example,

$$\text{AIC} = 2k - 2\log L$$

and

$$\text{BIC} = k\log n - 2\log L,$$

where k is the number of parameters, L is the maximised likelihood, and n is the sample size.

Cross-validation, by contrast, does not rely on such a closed-form penalty. In K-fold cross-validation, the dataset is split into K folds, and the model is trained K times, each time leaving out one fold for validation. If the loss on fold j is E_j, then the cross-validation estimate is

$$\text{CV error} = \frac{1}{K}\sum_{j=1}^{K} E_j.$$

So the two approaches differ both mathematically and conceptually. Information criteria use a theoretical correction to training fit; cross-validation estimates out-of-sample error directly through repeated resampling.

5.0.13 Worked Example

Worked Example

Suppose we are comparing two regression models on the same dataset.

Model A is simpler and has fewer parameters. Model B is more flexible and achieves a higher training likelihood.

An information criterion such as AIC may prefer Model B if the gain in likelihood is large enough to justify the added complexity. BIC may instead prefer Model A if the sample size is large and the stronger penalty for additional parameters outweighs the improvement in fit.

Now consider cross-validation. If Model B fits the training data better but does not generalise well, then its validation error may be worse than Model A's across held-out folds. In that case, cross-validation would prefer Model A.

This illustrates the central practical difference: information criteria infer the trade-off through a formula, while cross-validation tests generalisation more directly.

5.0.14 Python Demonstration

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, make_scorer

# Generate synthetic non-linear data
rng = np.random.default_rng(0)
X = np.linspace(-3, 3, 120)
y_true = 1.0 + 2.0 * X - 1.5 * X**2 + 0.4 * X**3
y = y_true + rng.normal(0, 3.0, size=len(X))

X = X.reshape(-1, 1)
n = len(X)

degrees = range(1, 9)
aic_values = []
bic_values = []
cv_mse_values = []

kf = KFold(n_splits=5, shuffle=True, random_state=0)
mse_scorer = make_scorer(mean_squared_error, greater_is_better=False)

for degree in degrees:
    # Fit on full data for AIC/BIC
    poly = PolynomialFeatures(degree=degree, include_bias=True)
    X_poly = poly.fit_transform(X)

    model = LinearRegression(fit_intercept=False)
    model.fit(X_poly, y)
    y_pred = model.predict(X_poly)

    residuals = y - y_pred
    rss = np.sum(residuals**2)
    sigma2 = rss / n
    logL = -n / 2 * (np.log(2 * np.pi * sigma2) + 1)

    k = X_poly.shape[1]

    aic = 2 * k - 2 * logL
    bic = k * np.log(n) - 2 * logL

    aic_values.append(aic)
    bic_values.append(bic)

```

```
    # Cross-validation estimate
    pipeline = Pipeline([
        ("poly", PolynomialFeatures(degree=degree, include_bias=True)),
        ("linreg", LinearRegression(fit_intercept=False))
    ])

    scores = cross_val_score(
        pipeline, X, y, cv=kf, scoring=mse_scorer
    )
    cv_mse_values.append(-scores.mean())

plt.show()
```

Listing 5.2: Comparing AIC, BIC, and cross-validation for polynomial regression models

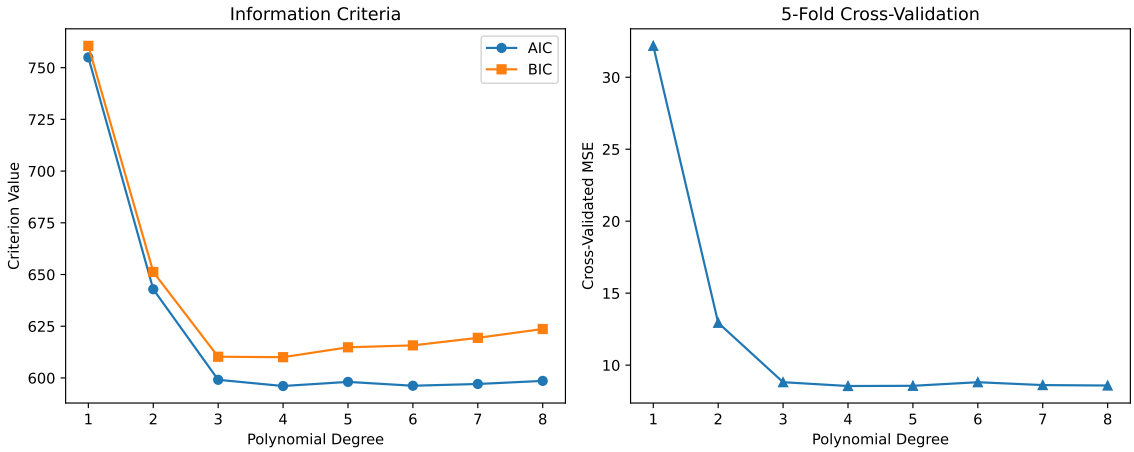


Figure 5.2: Comparison of AIC, BIC, and cross-validated error across polynomial model complexity. Information criteria combine fit with an explicit penalty for additional parameters, while cross-validation estimates predictive performance on held-out data. The preferred model may differ because the methods target related but not identical notions of model quality.

This example highlights the practical distinction between the two approaches. AIC and BIC can be computed from a single fitted model once the likelihood is available, so they are often much cheaper to evaluate than repeated cross-validation. Cross-validation, however, can provide a more direct estimate of predictive performance on unseen data, especially when the modelling assumptions behind likelihood-based criteria are questionable. In practice, it is not unusual for AIC, BIC, and cross-validation to prefer slightly different levels of model complexity.

5.0.15 Deep Dive: When Should You Prefer One Over the Other?

Deep Dive

Information criteria are often especially appealing in classical statistical modelling. If the model is likelihood-based, the assumptions are reasonably well matched to the problem, and the goal is to compare a finite set of candidate models, then AIC or BIC can be elegant and efficient tools. They are particularly common in linear models, generalised linear models, mixture models, and time series settings such as ARIMA selection.

Cross-validation is usually more attractive when prediction is the main objective, when model assumptions are weak or hard to justify, or when comparing methods that are not naturally likelihood-based. For example, if we are comparing tree ensembles, kernel methods, and neural networks, cross-validation provides a common predictive framework even though the notion of parameter counting or likelihood penalty may be unclear or not directly comparable.

There is also an important philosophical difference. AIC is often viewed as being more aligned with predictive accuracy, because of its connection to expected information loss. BIC is often viewed as being more conservative and more oriented toward selecting a simpler "true" model when such a notion is meaningful. Cross-validation, by contrast, usually avoids making this distinction explicit and focuses instead on empirical out-of-sample performance. A strong interview answer should therefore avoid claiming that one method is universally better. A more accurate view is that information criteria are fast, elegant, and theoretically motivated within likelihood-based modelling, while cross-validation is more flexible and often more directly relevant to predictive performance.

5.0.16 Follow-Up Interview Questions

Follow-Up Interview Questions

- Why might AIC and cross-validation favour similar models in some settings?
- Why is cross-validation often preferred in machine learning practice?
- When is BIC more conservative than AIC?
- Why can cross-validation be computationally expensive?
- Can information criteria be used for models without a clear likelihood?

5.0.17 Common Mistakes

Common Mistake

Saying that information criteria and cross-validation are identical tools. They serve similar purposes, but they estimate model quality in different ways.

Common Mistake

Assuming that information criteria always provide a better estimate of predictive performance than cross-validation.

Common Mistake

Ignoring the fact that AIC and BIC require a meaningful likelihood-based modelling framework.

Common Mistake

Claiming that cross-validation automatically solves all model-selection problems without considering variance, data size, or computational cost.

5.0.18 Summary

Information criteria and cross-validation are both useful tools for model selection, but they approach the problem from different directions. Information criteria such as AIC and BIC start from likelihood and then apply an explicit penalty for complexity, making them efficient and theoretically natural in statistical modelling. Cross-validation instead estimates out-of-sample performance directly by repeatedly evaluating the model on held-out data, which often makes it more relevant for predictive machine learning. In practice, the choice between them depends on the modelling framework, the computational budget, and whether the goal is parsimonious statistical explanation or strong predictive performance.

Chapter 6

Statistical Testing

Introduction

When comparing machine learning models, we often observe differences in performance metrics such as accuracy, F1 score, or RMSE. However, these differences may arise due to randomness in the data rather than true differences in model quality.

Statistical testing provides a framework for determining whether observed differences are **statistically significant**, meaning they are unlikely to have occurred by chance. Understanding statistical significance is essential for rigorous model evaluation and is a common topic in technical interviews.

Interview Question

Question 20: What is statistical significance in model evaluation?

Difficulty Level

Tier: Advanced

6.0.1 Short Interview Answer

Short Interview Answer

Statistical significance in model evaluation refers to determining whether the observed difference in performance between models is likely due to a real effect rather than random variation in the data.
This is typically assessed using hypothesis testing and p-values.

6.0.2 Intuition

When we evaluate models, performance varies depending on:

- how the data is split
- randomness in training
- sampling variability

So even if one model appears better, the key question is: *Is this difference real, or just noise?* Statistical significance helps answer this question.

Interview Tip

A strong answer should clearly distinguish between **observed difference** and **true difference**.

6.0.3 Hypothesis Testing Framework

Mathematical Insight

We define:

- Null hypothesis H_0: both models perform equally
- Alternative hypothesis H_1: one model performs better

We compute a test statistic and a p-value:

$$p = P(\text{observed difference} \mid H_0).$$

If p is small (e.g. < 0.05), we reject H_0.

6.0.4 Worked Example

Worked Example

Suppose we evaluate two models using cross-validation:

- Model A: accuracy = 0.85
- Model B: accuracy = 0.87

If the difference is consistent across folds and yields a low p-value, we conclude that Model B is significantly better.
If not, the difference may be due to randomness.

6.0.5 Python Demonstration

```python
import numpy as np
from scipy.stats import ttest_rel
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

# Generate data
X, y = make_classification(n_samples=300, n_features=10, random_state=42)

# Models
model1 = LogisticRegression(max_iter=1000)
model2 = DecisionTreeClassifier(max_depth=5)

# Cross-validation scores
scores1 = cross_val_score(model1, X, y, cv=5)
scores2 = cross_val_score(model2, X, y, cv=5)

# Paired t-test
```

```
t_stat, p_value = ttest_rel(scores1, scores2)

print("Scores model 1:", scores1)
print("Scores model 2:", scores2)
print(f"p-value: {p_value:.4f}")
```

Listing 6.1: Statistical test for model comparison

This test evaluates whether the difference in performance is statistically significant across folds.

6.0.6 Common Tests Used

- Paired t-test (most common with cross-validation)
- Wilcoxon signed-rank test (non-parametric alternative)
- Bootstrap methods

6.0.7 Deep Dive: Limitations of Statistical Testing

Deep Dive

Statistical tests in machine learning have important limitations:

- **Dependence between folds** Cross-validation folds are not fully independent, which can violate test assumptions.
- **Multiple comparisons** Comparing many models increases the chance of false positives.
- **Practical vs statistical significance** A statistically significant improvement may be too small to matter in practice.
- **Distributional assumptions** Tests like the t-test assume normality, which may not hold.

Because of these issues, statistical testing should be combined with domain knowledge and practical considerations.

6.0.8 Follow-Up Interview Questions

Follow-Up Interview Questions

- What does a p-value represent?
- Why do we use paired tests for model comparison?
- What are the limitations of hypothesis testing?
- What is the difference between statistical and practical significance?

6.0.9 Common Mistakes

Common Mistake

Assuming a higher metric always implies a better model without testing significance.

Common Mistake

Misinterpreting p-values as the probability that a model is correct.

Common Mistake

Ignoring practical significance in favour of statistical significance.

6.0.10 Summary

Statistical significance helps determine whether observed differences in model performance are likely to reflect genuine differences rather than random variation in the data or evaluation process. In practice, two models may produce slightly different scores simply because of sampling noise, train–test splits, or other sources of randomness, so hypothesis testing can be used to assess whether the observed gap is large enough to be considered statistically meaningful. P-values provide one way to quantify how plausible it would be to observe such a difference if there were no real underlying performance gap. However, statistical significance and practical significance are not the same: a difference can be statistically detectable yet too small to matter in a real application, while a practically important improvement may fail to reach significance if the sample is limited.

Interview Question

Question 21: What does a p-value represent?

Difficulty Level

Tier: Deep / Research Level

6.0.11 Short Interview Answer

Short Interview Answer

A p-value represents the probability of observing a result at least as extreme as the one obtained, assuming that the null hypothesis is true.
It does not measure the probability that the null hypothesis is true, but rather how consistent the observed data is with the null hypothesis.

6.0.12 Intuition

A p-value answers the question:

> If there were **no real difference between models**, how likely is it that we would observe a difference this large purely by chance?

- Small p-value → unlikely under the null → evidence against H_0
- Large p-value → plausible under the null → insufficient evidence

Interview Tip

A strong answer should clearly state that a p-value is **not** the probability that the null hypothesis is true.

6.0.13 Mathematical Definition

Mathematical Insight

Let T be a test statistic (e.g. difference in means).
The p-value is:

$$p = P(T \geq T_{\text{observed}} \mid H_0).$$

It measures how extreme the observed statistic is under the null hypothesis.

6.0.14 Worked Example

Worked Example

Suppose we compare two models and obtain a p-value of 0.03.
This means:

- If the models were actually equal, there is a 3% chance of observing a difference this large (or larger)

Since this is small, we reject the null hypothesis at the 5% level.

6.0.15 Python Demonstration

```python
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)

# Simulate differences under null (mean = 0)
differences = np.random.normal(0, 1, 1000)

# Observed difference
observed = 2.0

# Compute p-value (two-sided)
p_value = np.mean(np.abs(differences) >= observed)

print(f"Simulated p-value: {p_value:.3f}")

# Plot distribution
plt.figure(figsize=(6,5))
plt.hist(differences, bins=30, alpha=0.7, label="Null distribution")
plt.axvline(observed, color='red', linestyle='--', label="Observed")
plt.axvline(-observed, color='red', linestyle='--')
plt.title("P-value as Tail Probability")
plt.legend()
plt.tight_layout()
plt.show()
```

Listing 6.2: Simulating p-values under the null hypothesis

This visualisation shows that the p-value corresponds to the probability of observing extreme values under the null.

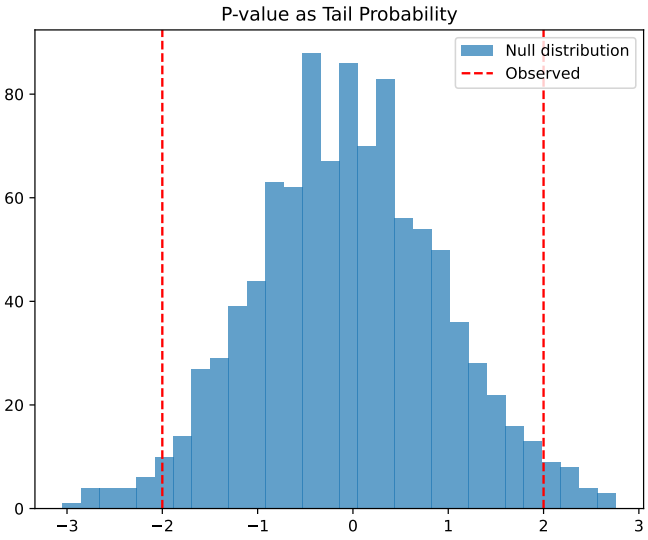


Figure 6.1: The p-value corresponds to the probability mass in the tails beyond the observed statistic under the null distribution.

6.0.16 Deep Dive: Common Misinterpretations

Deep Dive

P-values are frequently misunderstood. Common misconceptions include:

- **"p = 0.05 means a 5% chance the null is true"** Incorrect. The p-value assumes the null is true.
- **"A low p-value proves the alternative hypothesis"** Incorrect. It only provides evidence against the null.
- **"A high p-value proves no effect"** Incorrect. It may simply mean insufficient data.
- **Arbitrary thresholds (e.g. 0.05)** These are conventions, not universal truths.

In machine learning, where datasets are large, even very small effects can become statistically significant. Therefore, it is essential to consider **effect size** and **practical significance** alongside p-values.

6.0.17 Follow-Up Interview Questions

Follow-Up Interview Questions

- What are common misconceptions about p-values?
- What is the difference between statistical and practical significance?
- Why can large datasets produce small p-values?
- What alternatives exist to p-values?

6.0.18 Common Mistakes

Common Mistake

Interpreting the p-value as the probability that the null hypothesis is true.

Common Mistake

Treating p-values as definitive proof rather than evidence.

Common Mistake

Ignoring effect size and practical importance.

6.0.19 Summary

P-values are a key concept in statistical testing. Key ideas to remember for an interview include:

- They measure how extreme results are under the null hypothesis
- They do not represent the probability that the null is true
- Small p-values indicate evidence against the null
- They must be interpreted carefully in context

Interview Question

Question 22: What is the difference between one-tailed and two-tailed hypothesis tests?

Difficulty Level

Tier: Advanced

6.0.20 Short Interview Answer

Short Interview Answer

A one-tailed test evaluates whether a parameter deviates from the null hypothesis in a specific direction, while a two-tailed test evaluates whether it deviates in either direction. In a one-tailed test, the rejection region lies in one tail of the distribution, whereas in a two-tailed test, it is split across both tails.

6.0.21 Intuition

The key difference is **what kind of deviation from the null hypothesis we care about**.

- One-tailed test → we care about a difference in a specific direction
- Two-tailed test → we care about any difference (positive or negative)

For example:

- "Is model A better than model B?" → one-tailed
- "Are the models different?" → two-tailed

Interview Tip

A strong answer should explicitly mention that two-tailed tests check for **differences in both directions**.

A powerful way to understand hypothesis testing is to view it as a problem of separating two overlapping distributions:

- the distribution of a test statistic under the null hypothesis H_0
- the distribution under the alternative hypothesis H_1

Because these distributions overlap, no decision rule can perfectly separate them. Instead, we choose a **threshold (critical value)**:

- values beyond the threshold → reject H_0

- values within the threshold → do not reject H_0

The difference between one-tailed and two-tailed tests is how this rejection region is defined.

Interview Tip

A strong answer should explain that hypothesis testing is about **separating overlapping distributions** using a threshold.

6.0.22 Mathematical Formulation

Mathematical Insight

Let θ be a parameter.

Two-tailed test:

$$H_0 : \theta = \theta_0, \quad H_1 : \theta \neq \theta_0$$

One-tailed test (right):

$$H_0 : \theta \leq \theta_0, \quad H_1 : \theta > \theta_0$$

One-tailed test (left):

$$H_0 : \theta \geq \theta_0, \quad H_1 : \theta < \theta_0$$

6.0.23 Visual Explanation

The figure below shows the distributions of a test statistic under H_0 and H_1. Because they overlap, some outcomes are ambiguous.

Interpretation:

- Right of threshold → reject H_0
- Under H_0: this region corresponds to **Type I error**
- Under H_1: this region corresponds to **power**
- Left of threshold under H_1: **Type II error**

6.0.24 One-Tailed vs Two-Tailed Tests

One-tailed test:

- Rejection region lies entirely in one tail
- Used when only one direction matters
- More sensitive in that direction

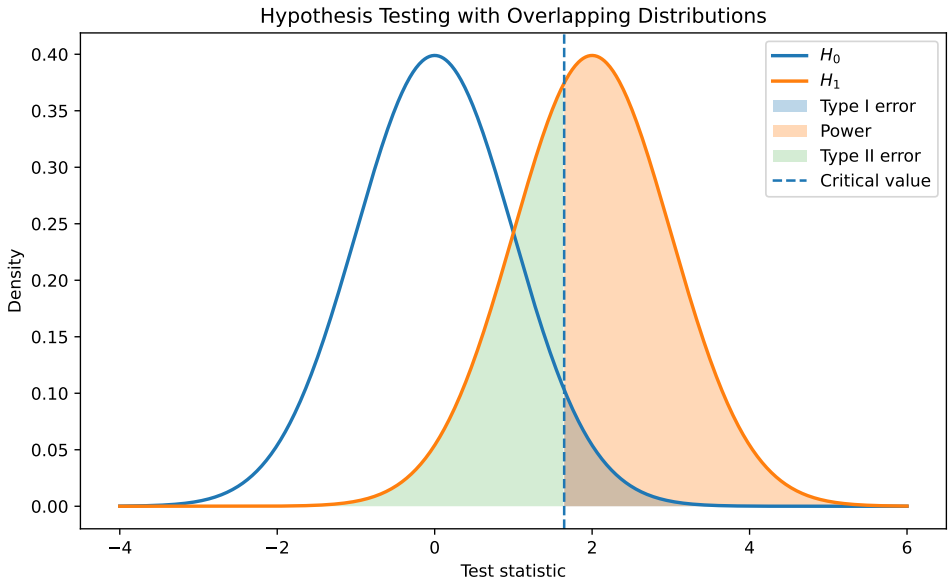


Figure 6.2: Overlapping distributions under H_0 and H_1. The vertical line is the critical value. The shaded regions illustrate Type I error, Type II error, and statistical power.

Two-tailed test:

- Rejection region split across both tails
- Detects deviations in either direction
- More conservative

6.0.25 Worked Example

Worked Example

Suppose we are testing whether a new model improves performance.

- If we only care about improvement → one-tailed test
- If we care about any difference → two-tailed test

A two-tailed test will reject the null if the new model is either significantly better or significantly worse.

6.0.26 Python Demonstration

This python code can be used to generate Figure 6.2.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

```

```
x = np.linspace(-4, 6, 1000)

# Distributions
h0 = norm.pdf(x, loc=0, scale=1)
h1 = norm.pdf(x, loc=2, scale=1)

critical_value = 1.645  # right-tailed test

plt.figure(figsize=(8,5))
plt.plot(x, h0, label=r"$H_0$")
plt.plot(x, h1, label=r"$H_1$")

# Type I error
plt.fill_between(x, h0, where=(x >= critical_value), alpha=0.3, label="Type I error")

# Power
plt.fill_between(x, h1, where=(x >= critical_value), alpha=0.3, label="Power")

# Type II error
plt.fill_between(x, h1, where=(x < critical_value), alpha=0.2, label="Type II error")

plt.axvline(critical_value, linestyle="--", label="Critical value")

plt.title("Hypothesis Testing via Overlapping Distributions")
plt.xlabel("Test statistic")
plt.ylabel("Density")
plt.legend()
plt.tight_layout()
plt.show()
```

Listing 6.3: Overlapping distributions for hypothesis testing

6.0.27 Deep Dive: Trade-Offs and Power

Deep Dive

The overlap between H_0 and H_1 explains why hypothesis testing involves trade-offs:

- Reducing Type I error (false positives) increases Type II error
- Increasing power often increases false positives

These trade-offs depend on:

- effect size (distance between distributions)

- sample size
- variance
- choice of threshold

One-tailed tests concentrate the rejection region in one direction, increasing power for detecting directional effects.
Two-tailed tests split the rejection region, making them more conservative but more general.

6.0.28 Follow-Up Interview Questions

Follow-Up Interview Questions

- What are Type I and Type II errors?
- What is statistical power?
- When is a one-tailed test appropriate?
- Why are two-tailed tests more commonly used?

6.0.29 Common Mistakes

Common Mistake
Using a one-tailed test without a justified directional hypothesis.

Common Mistake
Thinking one-tailed tests are always preferable due to higher power.

Common Mistake
Failing to connect hypothesis testing to overlapping distributions.

6.0.30 Summary

One-tailed and two-tailed tests differ in how they define the rejection region.

Key ideas include:

- Hypothesis testing can be viewed as separating overlapping distributions
- One-tailed tests focus on a single direction
- Two-tailed tests detect deviations in both directions
- Trade-offs arise due to unavoidable overlap between distributions

Part IV

Conceptual Foundations

In this part of the book, we move beyond algorithms and evaluation techniques to explore the statistical and theoretical foundations of machine learning. These concepts provide the underlying principles that explain why our models work, when they fail, and how their behaviour can be interpreted.

Understanding these ideas is essential for developing a deeper intuition about machine learning methods. It allows you to reason about uncertainty, make principled decisions about model design, and recognise the limitations of common approaches.

Many of the techniques used throughout earlier chapters—such as loss functions, regularisation, and model evaluation—are rooted in these fundamental concepts. As a result, this section not only strengthens your theoretical understanding but also enhances your ability to apply machine learning effectively in practice.

Chapter 7

Maximum Likelihood Estimation

Introduction

Maximum Likelihood Estimation (MLE) is one of the most fundamental concepts in statistics and plays a central role in machine learning. Many commonly used models and loss functions—such as linear regression, logistic regression, and log loss—can be derived from the principle of maximum likelihood.

Understanding MLE provides a unifying framework for interpreting model training as a probabilistic optimisation problem, making it a very common topic in technical interviews.

Interview Question

Question 23: What is maximum likelihood estimation?

Difficulty Level

Tier: Core Question

7.0.1 Short Interview Answer

Short Interview Answer

Maximum likelihood estimation is a method for estimating model parameters by choosing the values that maximise the likelihood of observing the given data under the model.

7.0.2 Intuition

MLE is based on a simple idea:

> Choose the parameters that make the observed data most likely.

Instead of asking:

> "What parameters could have generated this data?"

we ask:

> "Which parameters make this data most probable?"

This turns model fitting into an optimisation problem.

Interview Tip

A strong answer should clearly state that MLE chooses parameters that **maximise the likelihood of the observed data**.

7.0.3 Mathematical Definition

Mathematical Insight

Let $x_1, x_2, \ldots, x_n$ be independent observations with probability density $p(x \mid \theta)$.
The likelihood function is:

$$L(\theta) = \prod_{i=1}^{n} p(x_i \mid \theta).$$

MLE chooses:

$$\hat{\theta} = \arg\max_{\theta} L(\theta).$$

In practice, we maximise the log-likelihood:

$$\log L(\theta) = \sum_{i=1}^{n} \log p(x_i \mid \theta).$$

7.0.4 Worked Example

Worked Example

Suppose data is generated from a normal distribution:

$$x_i \sim \mathcal{N}(\mu, \sigma^2).$$

The MLE estimate of the mean is:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

Thus, the sample mean is the maximum likelihood estimator of μ.

7.0.5 Python Demonstration

```python
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)

# Generate data
data = np.random.normal(loc=5, scale=2, size=100)

# MLE estimate of mean
mu_hat = np.mean(data)

print(f"Estimated mean (MLE): {mu_hat:.2f}")

# Visualise likelihood for different means
mu_values = np.linspace(3, 7, 100)
likelihoods = []

for mu in mu_values:
```

```
    likelihood = np.prod(
        (1 / np.sqrt(2 * np.pi * 2**2)) *
        np.exp(-(data - mu)**2 / (2 * 2**2))
    )
    likelihoods.append(likelihood)

plt.figure(figsize=(6,5))
plt.plot(mu_values, likelihoods)
plt.axvline(mu_hat, linestyle='--', label="MLE estimate")
plt.xlabel("Mean parameter")
plt.ylabel("Likelihood")
plt.title("Likelihood Function")
plt.legend()
plt.tight_layout()
plt.show()
```

Listing 7.1: MLE for estimating a mean

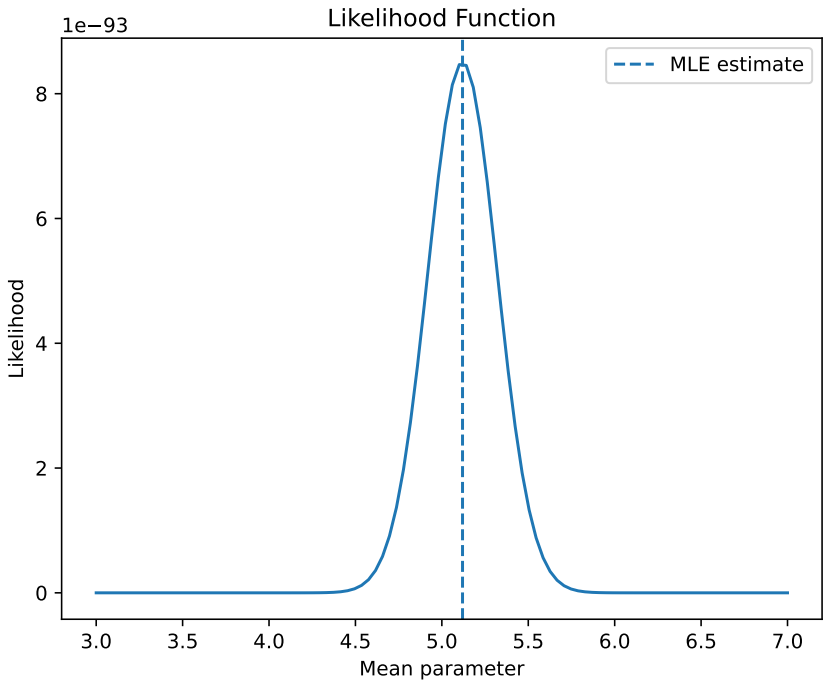


Figure 7.1: The likelihood function is maximised at the MLE estimate of the parameter.

7.0.6 Connection to Machine Learning

MLE provides a unifying view of many models:

- Linear regression → minimising MSE corresponds to MLE under Gaussian noise
- Logistic regression → minimising log loss corresponds to MLE under Bernoulli likelihood

Thus, many loss functions arise naturally from likelihood assumptions.

7.0.7 Deep Dive: Why Log-Likelihood Is Used

Deep Dive

The likelihood function involves a product of probabilities:

$$L(\theta) = \prod p(x_i \mid \theta).$$

This can become numerically unstable for large datasets.
Taking logs gives:

$$\log L(\theta) = \sum \log p(x_i \mid \theta),$$

which:

- avoids numerical underflow
- simplifies optimisation
- converts products into sums

This is why most machine learning models optimise log-likelihood rather than likelihood directly.

7.0.8 Follow-Up Interview Questions

Follow-Up Interview Questions

- How does MLE relate to loss functions?
- Why do we use log-likelihood instead of likelihood?
- What assumptions are required for MLE?
- How does MLE relate to Bayesian inference?

7.0.9 Common Mistakes

Common Mistake

Confusing likelihood with probability of parameters.

Common Mistake

Failing to explain why we take the logarithm.

Common Mistake

Not connecting MLE to machine learning loss functions.

7.0.10 Summary

Maximum likelihood estimation is a fundamental principle in statistics and machine learning because it chooses parameter values that make the observed data as probable as possible under the model. This turns model fitting into a clear optimisation problem, where the goal is to maximise the likelihood, or equivalently to minimise the negative log-likelihood. Many widely used loss functions can be understood in this way, which gives them a strong probabilistic foundation rather than viewing them as arbitrary optimisation choices. As a result, maximum likelihood estimation provides a powerful interpretation of learning: training a model can be seen as finding the parameters under which the observed data are most plausibly generated.

Chapter 8

Generative vs Discriminative Models

Introduction

In machine learning, models can be broadly categorised based on how they approach the problem of learning from data.

Some models focus on modelling the joint distribution of the data, while others focus directly on predicting the target variable. These two perspectives give rise to **generative** and **discriminative** models.

Understanding the difference between these approaches is important not only for theoretical clarity, but also for choosing appropriate models in practice.

Interview Question

Question 24: What is generative modelling?

Difficulty Level

Tier: Core Question

8.0.1 Short Interview Answer

Short Interview Answer

Generative modelling involves learning the joint distribution $p(x, y)$ or the data distribution $p(x)$, allowing the model to generate new samples and perform probabilistic inference.

8.0.2 Intuition

A generative model tries to understand *how* the data was generated. Rather than focusing only on predicting labels, it learns a model of the underlying data distribution. This allows it to answer more flexible questions, such as:

- What does a typical data point look like?
- How likely is this observation?
- Can we generate new realistic samples?

For example, in a classification setting, a generative model learns how each class produces data. Once this is known, it can use Bayes' theorem to compute the probability of each class given an input.

Interview Tip

A strong answer should emphasise that generative models learn **how the data is generated**, not just how to predict labels.

Interview Tip

A strong answer should emphasise that generative models learn **how the data is generated**, not just how to predict labels.

8.0.3 Mathematical Perspective

Mathematical Insight

Generative models learn:

$$p(x, y) = p(x \mid y)p(y).$$

Predictions are made using Bayes' theorem:

$$p(y \mid x) = \frac{p(x \mid y)p(y)}{p(x)}.$$

8.0.4 Worked Example

Worked Example

Consider spam classification.
A generative model such as Naive Bayes learns:

- how likely certain words are in spam emails
- how likely they are in non-spam emails

Given a new email, the model computes the likelihood under each class and uses Bayes' theorem to determine the most probable class.

8.0.5 When to Use Generative Models

Generative models are particularly useful when:

- we want to generate new data (e.g. images, text)
- we need to model uncertainty explicitly
- data is limited, and strong assumptions help generalisation
- we want to handle missing data naturally

For example, generative models are widely used in image generation (e.g. GANs), anomaly detection, and probabilistic modelling.

8.0.6 Deep Dive: Advantages and Trade-Offs

Deep Dive

Because generative models learn the full data distribution, they are more flexible and can be used for a wider range of tasks beyond prediction.
However, this flexibility comes at a cost. Modelling the full distribution $p(x, y)$ is often more complex than modelling $p(y \mid x)$, and may require stronger assumptions.
In practice, generative models may underperform discriminative models in pure prediction tasks, especially when large amounts of labelled data are available.

8.0.7 Follow-Up Interview Questions

Follow-Up Interview Questions

- What is discriminative modelling?
- What are examples of generative models?
- When would you choose a generative model over a discriminative one?

8.0.8 Common Mistakes

Common Mistake

Saying generative models are only used for data generation.

Common Mistake

Failing to explain the role of $p(x \mid y)$ and Bayes' theorem.

8.0.9 Summary

Generative models learn how data is produced by modelling the joint distribution. This enables both prediction and data generation, making them flexible but often more complex than discriminative approaches.

Interview Question

Question 25: What is discriminative modelling?

Difficulty Level

Tier: Core Question

8.0.10 Short Interview Answer

Short Interview Answer

Discriminative modelling focuses on learning the conditional probability $p(y \mid x)$ or directly learning a decision boundary to predict labels from inputs.

8.0.11 Intuition

A discriminative model focuses directly on the task of prediction*prediction*. Rather than modelling how the data is generated, it learns how to distinguish between different classes. In other words, it focuses on learning the boundary that separates classes in the feature space.

This makes discriminative models more efficient for prediction tasks, since they do not attempt to model unnecessary aspects of the data.

Interview Tip

A strong answer should emphasise that discriminative models focus on **decision boundaries or conditional probabilities**.

8.0.12 Mathematical Perspective

Mathematical Insight

Discriminative models learn:

$$p(y \mid x)$$

or directly learn a function:

$$f(x) \to y.$$

8.0.13 Worked Example

Worked Example

In spam classification, a discriminative model such as logistic regression learns a decision boundary that separates spam from non-spam emails.
Rather than modelling how each type of email is generated, it focuses only on distinguishing between the two classes.

8.0.14 When to Use Discriminative Models

Discriminative models are typically preferred when:

- the goal is accurate prediction
- large amounts of labelled data are available
- the data distribution is complex and difficult to model

Examples include logistic regression, support vector machines, and neural networks.

8.0.15 Deep Dive: Generative vs Discriminative Trade-Off

Deep Dive

The choice between generative and discriminative models depends on the problem setting. Discriminative models tend to achieve better predictive performance when sufficient labelled data is available, because they focus directly on the task of interest.
Generative models, on the other hand, can be more robust when data is scarce or when we need to model uncertainty or generate new data.
This leads to a useful rule of thumb:

> Use discriminative models for prediction, and generative models for understanding and generation.

However, modern machine learning increasingly blends these approaches, particularly in deep learning.

8.0.16 Follow-Up Interview Questions

Follow-Up Interview Questions

- How do generative and discriminative models differ?
- Why might discriminative models perform better in practice?
- Can you combine both approaches?

8.0.17 Common Mistakes

Common Mistake

Describing discriminative models without contrasting them with generative models.

Common Mistake

Assuming discriminative models cannot model uncertainty.

8.0.18 Summary

Discriminative models focus on predicting labels directly by modelling $p(y \mid x)$ or learning decision boundaries. They are often preferred for prediction tasks due to their simplicity and strong performance with sufficient data.

Part V

Applied ML and Systems

In the previous parts of this book, we focused on the theory and core techniques of machine learning. We explored how models are built, trained, and evaluated, as well as the statistical foundations that underpin them. In this part, we shift our focus to how machine learning is used in practice.

Real-world machine learning systems are not just localised models. They are pipelines, systems, and processes that must operate reliably at scale. They must handle messy data, adapt to changing environments, and integrate with production systems. All of this introduces a new set of challenges:

- dealing with imperfect and evolving data
- deploying models into production environments
- monitoring performance over time
- designing systems that scale and remain reliable

Understanding these aspects is essential for applied data science and machine learning roles. In many interviews, candidates are evaluated not just on their knowledge of models, but on their ability to reason about real-world systems. This part of the book is therefore focused on bridging the gap between theory and practice.

Chapter 9

Production ML Systems

Introduction

Once a model has been trained and deployed, it must operate as part of a larger production system. At this stage, the main challenge is no longer just predictive performance, but how predictions are delivered reliably, efficiently, and at the right time.

Production machine learning systems must handle live requests, process large volumes of data, meet latency requirements, and integrate with other software components. This requires careful design of the inference layer and a clear understanding of how predictions are generated in practice.

Questions about model serving and inference patterns are common in applied machine learning and system design interviews. Even if you have had no prior experience with production systems, some awareness about what is involved will be crucial to passing the technical interviews.

Interview Question

Question 26: What is model serving?

Difficulty Level

Tier: Core Question

9.0.1 Short Interview Answer

Short Interview Answer

Model serving is the process of making a trained machine learning model available for inference in a production environment, so that it can receive input data and return predictions to other systems or users.

9.0.2 Intuition

A trained model is only useful in production if other systems can actually call it and obtain predictions. Model serving is the layer that makes this possible. In practice, this usually means placing the model inside an application, service, or endpoint that accepts input data, applies the necessary preprocessing steps, runs inference, and returns the output in a form that another system can use.

For example, an e-commerce website may call a recommendation model to generate product suggestions. A payment system may call a fraud model before approving a transaction. In both cases, the model is not being used manually by a data scientist. It's being served as part of an operational system. This leads to a useful way of thinking about it:

> Model serving is the bridge between a trained model and the real-world application that depends on its predictions.

Interview Tip

A strong answer should make clear that model serving is about **operational access to inference**, not just storing a model file.

9.0.3 What a Serving System Typically Does

A model serving system usually does more than just run the model itself. It often includes input validation, feature transformation, model loading, prediction logic, logging, and error handling. In some systems, it may also manage model versions, perform A/B routing, or fall back to an earlier model if something fails.

This means that serving is not simply the last line of code calling `predict()`. It is a **production component** that must be reliable, observable, and maintainable.

9.0.4 Worked Example

Worked Example

Suppose a company has trained a churn prediction model.
The model is deployed behind an internal API. When the customer success platform requests a churn score for a user, the serving system receives the input features, applies the same preprocessing used during training, loads the correct model version, and returns a probability of churn.
That score is then used to decide whether to trigger a retention action.
In this setup, the serving layer is what allows the trained model to become part of a real business workflow.

9.0.5 Python Demonstration

```python
import numpy as np
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression

# Train a simple model
X, y = make_classification(n_samples=200, n_features=5, random_state=0)
model = LogisticRegression(max_iter=1000)
model.fit(X, y)

# Simulated serving function
def serve_prediction(input_features):
    input_array = np.array(input_features).reshape(1, -1)
    probability = model.predict_proba(input_array)[0, 1]
    prediction = model.predict(input_array)[0]
    return {"prediction": int(prediction), "probability": float(probability
        )}

# Example request
result = serve_prediction([0.1, -1.2, 0.4, 0.7, -0.3])
print(result)
```

Listing 9.1: Simple model serving style example

This example is much simpler than a real production system, but it illustrates the core serving pattern: accept input, run inference, and return an output.

9.0.6 Deep Dive: Serving Is a Reliability Problem

Deep Dive

A useful production perspective is that model serving is as much a reliability problem as it is a modelling problem.
A highly accurate model is not useful if it cannot respond within the required latency, crashes under load, or produces inconsistent results because preprocessing differs between training and inference.
For this reason, production serving systems must be designed with concerns such as:

- latency
- throughput
- versioning
- reproducibility
- logging and monitoring

In many cases, these concerns determine whether a model is viable in production at all.

9.0.7 Follow-Up Interview Questions

Follow-Up Interview Questions

- What is batch inference?
- What is real-time inference?
- How do you avoid training-serving skew?
- What should be monitored in a serving system?

9.0.8 Common Mistakes

Common Mistake

Describing model serving as simply saving a model to disk.

Common Mistake

Ignoring preprocessing and infrastructure around inference.

Common Mistake

Focusing only on predictive accuracy instead of system requirements such as latency and reliability.

9.0.9 Summary

Model serving is the process of making a trained model available for production inference. It turns a model into a usable system component by accepting inputs, generating predictions, and integrating those predictions into a larger application reliably and at scale.

Interview Question

Question 27: What is batch vs real-time inference?

Difficulty Level

Tier: Core Question

9.0.10 Short Interview Answer

Short Interview Answer

Batch inference generates predictions for many examples at scheduled intervals, while real-time inference generates predictions on demand for individual requests or small groups of requests with low latency.

9.0.11 Intuition

The difference between batch and real-time inference is mainly about **when** predictions are made and **how quickly** they are needed. In some applications, predictions do not need to be generated instantly. It is acceptable to score many records together at regular intervals, such as once per hour or once per day. This is batch inference.

In other applications, predictions are needed immediately in response to a user action or system event. For example, a fraud model may need to decide whether to block a payment before the transaction completes. This is real-time inference. The core distinction is therefore:

> Batch inference optimises for throughput, while real-time inference optimises for latency.

Interview Tip

A strong answer should explain the trade-off between **throughput** and **latency**.

9.0.12 Batch Inference

In batch inference, predictions are generated for many examples at once. This is common when predictions can be prepared in advance and stored for later use. For example, a recommendation system might precompute product suggestions overnight, or a churn model might score all customers once per week.

Batch inference is often simpler and cheaper to operate because it does not require low-latency serving for each individual request. It is also well suited to large-scale offline pipelines. However, its main limitation is that predictions may become stale between runs.

9.0.13 Real-Time Inference

In real-time inference, the model is called on demand. This is necessary when predictions depend on the latest user state or when the result must be returned immediately. Search ranking, fraud detection, dynamic pricing, and conversational AI are all examples where real-time inference is important.

Real-time systems must meet strict latency requirements and be highly available. This makes them more difficult to build and maintain than batch systems. The benefit, however, is that predictions can reflect the most recent data and be used immediately in interactive systems.

9.0.14 Worked Example

Worked Example

Consider an online retailer.
A recommendation model used for the homepage might precompute recommended products every night. This is batch inference.
A fraud detection model used during payment processing must evaluate each transaction instantly before it is approved. This is real-time inference.
Both are forms of inference, but the operational requirements are completely different.

9.0.15 Python Demonstration

```python
import numpy as np
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression

# Train model
X, y = make_classification(n_samples=200, n_features=5, random_state=0)
model = LogisticRegression(max_iter=1000)
model.fit(X, y)

# Batch inference: many examples at once
batch_inputs = X[:5]
batch_predictions = model.predict(batch_inputs)
print("Batch predictions:", batch_predictions)

# Real-time inference: one request at a time
single_input = X[5].reshape(1, -1)
single_prediction = model.predict(single_input)
print("Real-time prediction:", single_prediction[0])
```

Listing 9.2: Batch vs real-time inference illustration

This example uses the same trained model in two different inference modes: many inputs processed together, and one input processed on demand.

9.0.16 Deep Dive: Choosing Between Batch and Real-Time

Deep Dive

The choice between batch and real-time inference depends on the application rather than the model itself.

If predictions can be prepared ahead of time and updated periodically, batch inference is often simpler, cheaper, and more robust. If predictions must react to fresh data or support interactive decisions, real-time inference is necessary.

In practice, many production systems use a hybrid design. Some features or predictions are precomputed in batch, while other signals are incorporated at request time. This allows the system to balance freshness, cost, and latency.

9.0.17 Follow-Up Interview Questions

Follow-Up Interview Questions

- When would batch inference be preferable to real-time inference?
- Why is real-time inference harder to operate?
- What are the latency challenges in online serving?
- Can a production system use both batch and real-time inference?

9.0.18 Common Mistakes

Common Mistake

Thinking that real-time inference is always better than batch inference.

Common Mistake

Ignoring the operational cost and complexity of low-latency serving.

Common Mistake

Describing the difference only in terms of dataset size rather than timing and system requirements.

9.0.19 Summary

Batch and real-time inference are two common ways of serving machine learning predictions. Batch inference generates predictions on a schedule and is well suited to offline workloads, while real-time inference generates predictions on demand and is essential for interactive or time-sensitive systems.

Index

AIC, 92
alternative hypothesis, 113

bias, 12
bias–variance trade-off, 12, 19
BIC, 92

class labels, 65
classification, 65
Cross-Entropy, 31
cross-entropy, 49, 57

discriminative, 126, 130

Gaussian distribution, 40
generative, 126
gradient descent, 31, 69, 75, 82

hypothesis testing, 114

inference, 140
Information criteria, 92
interviews, 2

KL divergence, 53

learning rate, 72, 77
likelihood, 40, 66
linear regression, 35
log-loss, 57
logistic regression, 35
loss function, 31, 65, 70, 82

Maximum Likelihood Estimation, 120
maximum likelihood estimation, 40, 66
mean squared error, 36
mini-batch gradient descent, 89
MLE, 120
model serving, 136
MSE, 36
multi-class classification, 66

null hypothesis, 113

one-tailed test, 113
optimisation, 61, 69
overfitting, 12

p-value, 105, 109
polynomial regression, 21
probabilistic machine learning, 53
production, 135

robust regression, 35

statistical significance, 104
stochastic gradient descent, 83, 84

technical interviews, 2
two-tailed, 113
Type I error, 114
Type II error, 114

underfitting, 12

variance, 12