WEEK 1: Python Programming Basics

- Day 1 Introduction to Python and Development Setup
- Day 2 Control Flow in Python
- Day 3 Functions and Modules
- Day 4 Data Structures (Lists, Tuples, Dictionaries, Sets)
- Day 5 Working with Strings
- Day 6 File Handling
- Day 7 Pythonic Code and Project Work

Day

01

Introduction to Python and Development Setup

Overview of Python and its Role in AI

- Why Python for AI
- Applications of Python in AI
  - Machine Learning
  - Deep Learning
  - Data Science and Analytics
  - Natural Language Processing (NLP)
  - Automation and Scripting

Installing Python and Setting Up a Coding Environment

- Installing Python
- Setting Up a Coding Environment
  - Jupyter Notebooks
  - Visual Studio Code

Introduction to Basic Python Syntax, Variables, and Data Types

- Basic Syntax
- Variables
- Data Types
  - Integers | Floats | Strings | Lists | Tuples | Dictionaries | Booleans
- Examples of Using Variables and Data Types

Hands-On Exercise

- Exercise 1: Write a Python script to print a message
- Exercise 2: Practice Creating and Manipulating Variables of Different Data Types

Day

02

Control Flow in Python

Conditional Statements

- Syntax:
  - if: Executes code if a condition is True
  - elif: Adds additional conditions after the initial if
  - else: Executes code if none of the previous conditions are met

Loops

- for Loop
  - Iterates over a sequence
- while Loop
  - Executes as long as a condition is True

Using break and continue for Control Flow

- break
  - Terminates the loop prematurely when a condition is met
- continue
  - Skips the current iteration and proceeds to the next

Hands-On Exercise

- Exercise 1: Check if a Number is Prime
- Exercise 2: Create a Menu-Driven Calculator
- Additional Practice:
  - Write a program to calculate the factorial of a number using a while loop
  - Create a program to find the largest number in a list using a for loop

Day

03

Functions and Modules in Python

Defining Functions with def

- What are Functions?
- Defining a Function
- Examples

Scope and Lifetime of Variables

- Scope
  - Local Scope
  - Global Scope
- Lifetime
- Examples

Importing and Using Modules

- What are Modules?
- Importing Modules
  - Import an entire module
  - Import specific functions
  - Use aliases
- Creating Custom Modules

Hands-On Exercises

- Exercise 1: Create a Function to Calculate Factorials
- Exercise 2: Build a Custom Python Module
- Additional Practice:
  - Exercise 1: Write a function to check if a number is even or odd and call it within another function
  - Exercise 2: Create a module for string operations, including functions to reverse a string, count vowels, and check for palindromes. Import it into a script and test the functions

Day

04

Data Structures
(Lists, Tuples, Dictionaries, Sets)

Lists

- What are Lists?
- Creating Lists
- Accessing Elements
  - Access by index
  - Negative indexing
- Modifying Lists
- Slicing Lists

Tuples

- What are Tuples?
- Creating Tuples
- Accessing Elements
- Immutability

Dictionaries

- What are Dictionaries?
- Creating Dictionaries
- Accessing and Modifying Data
  - Access
  - Add or Update
  - Remove
- Iteration

Sets

- What are Sets?
- Creating Sets
- Adding and Removing Elements
- Set Operations
  - Union
  - Intersection
  - Difference

Hands-On Exercises

- Exercise 1: Manipulate Data in a Dictionary
- Exercise 2: Word Frequency Counter
- Additional Practice
  - Write a program to reverse a list and remove duplicates using a set
  - Create a program that stores student grades in a dictionary and calculates the average grade

Day

05

Working with Strings

String Manipulation

- Concatenation
- Slicing
- Formatting

Common String Methods

- split()
- join()
- replace()
- strip()

Regular Expressions for Pattern Matching

- What are Regular Expressions?
- Using the re Module
- Common Functions
  - re.search(pattern, string)
  - re.findall(pattern, string)
  - re.sub(pattern, replacement, string)
- Examples

Hands-On Exercises

- Exercise 1: Create a Text Cleaner
- Exercise 2: Check if a String is a Palindrome
- Additional Practice
  - Write a program to count the number of vowels in a string
  - Create a program to find and replace all email addresses in a text using regex
  - Write a program to reverse the words in a sentence (not the letters)

Day

06

File Handling

Reading and Writing Text Files

- Opening Files
  - Use the built-in open() function to open a file
    - r | w | a | r+
- Reading Files
  - .read() | .readline() | .readlines()
- Writing to Files
  - .write() | .writelines()

Using *'with'* Statements for File Management

- What is the with Statement?
  - Ensures files are properly closed after operations, even if an exception occurs
- Advantages
  - Simplifies code
  - Reduces the risk of file corruption

Basic Exception Handling for File Operations

- Why Use Exception Handling?
  - Prevents the program from crashing due to file-related errors (e.g., file not found)
- Using *try-except* Blocks
- Common File Handling Exceptions
  - FileNotFoundError
  - PermissionError
  - IOError

Hands-On Exercises

- Exercise 1: Count Words and Lines in a File
- Exercise 2: Write and Read a List of Items
- Additional Practice
  - Write a program to copy the contents of one file to another
  - Create a program that counts the number of occurrences of a specific word in a text file
  - Write a program to log messages with timestamps into a file

Day

07

Pythonic Code and Project Work

Writing Clean, “Pythonic” Code

- What is Pythonic Code?
- Best Practices
  - Use descriptive variable names
  - Write modular code with functions and classes
  - Follow PEP 8 style guidelines
  - Avoid redundancy; leverage Python’s powerful built-ins

List Comprehensions

- What Are List Comprehensions?
  - A concise way to create lists using a single line of code
- Examples

Lambda Functions

- What Are Lambda Functions?
  - Anonymous, single-expression functions defined using the lambda keyword
- Examples

map(), filter(), and reduce()

- map()
  - Applies a function to each item in an iterable
- filter()
  - Filters items based on a condition
- reduce()
  - Reduces an iterable to a single value

Python's *os* and *sys* Modules

- os Module
  - Provides functions to interact with the operating system
- sys Module
  - Provides access to system-specific parameters and functions

Hands-On Project

- Command-Line Task Manager
- Additional Features to Add
  - Add deadlines or priorities for tasks
  - Export tasks to a JSON or CSV file
  - Support command-line arguments for quicker task management

WEEK 2: Data Science Essentials

- Day 1 Introduction to NumPy for Numerical Computing
- Day 2 Advanced NumPy Operations
- Day 3 Introduction to Pandas for Data Manipulation
- Day 4 Data Cleaning and Preparation with Pandas
- Day 5 Data Aggregation and Grouping in Pandas
- Day 6 Data Visualization with Matplotlib and Seaborn
- Day 7 Exploratory Data Analysis (EDA) Project

Day

01

Introduction to NumPy for Numerical Computing

Understanding the Role of NumPy in Data Science and AI

- What is NumPy?
- Why Use NumPy in AI?
  - Performance
  - Ease of Use
  - Integration

Creating and Manipulating NumPy Arrays

- Import NumPy
- Creating Arrays
  - From a list
  - Using built-in functions
- Manipulating Arrays
  - Change shape
  - Add dimensions

Basic Operations on Arrays

- Element-wise Operations
- Mathematical Operations

Array Indexing, Slicing, and Reshaping

- Indexing
- Slicing
- Reshaping

Hands-On Exercises

- Exercise 1: Generate Arrays for Basic Mathematical Operations
- Exercise 2: Create a 3x3 Matrix and Perform Operations
- Additional Practice
  - Create a 4x4 matrix and calculate the sum of its rows and columns
  - Write a program to normalize an array (scale values between 0 and 1)
  - Generate a random array and find the minimum and maximum values

Day

02

Advanced NumPy Operations

Broadcasting in NumPy

- What is Broadcasting?
- Rules of Broadcasting
  - Dimensions are aligned from the right
  - A dimension is compatible if:
    - It matches the other array's dimension
    - One of the dimensions is 1

Aggregation Functions

- Aggregation functions compute summary statistics for arrays
- Common Functions

Boolean Indexing and Filtering

- What is Boolean Indexing?
- Examples

Random Number Generation and Setting Seeds

- Random Number Generation
  - np.random
- Setting Random Seeds

Hands-On Exercises

- Exercise 1: Broadcasting Operations
- Exercise 2: Generate and Filter a Random Dataset
- Additional Practice
  - Create a 3D random array and compute statistics along specific axes
  - Write a program to generate a dataset of random floats and normalize the values between 0 and 1
  - Implement conditional replacement to create a binary mask for values above a threshold

Day

03

Introduction to Pandas for Data Manipulation

Introduction to Pandas Data Structures

- What is Pandas?
- Pandas Data Structures
  - Series
  - DataFrame

Loading Data from CSV, Excel, and Other Sources

- Common Data Loading Methods
  - From CSV
  - From Excel
  - From a Dictionary
- Saving Data
  - Save to CSV
  - Save to Excel

Basic DataFrame Operations

- Viewing Data
- Selecting and Indexing
  - Selecting columns
  - Filtering rows
  - Selecting by position
  - Selecting by label

Hands-On Exercises

- Exercise 1: Load and Explore a Sample Dataset
- Exercise 2: Select Specific Columns and Filter Rows
- Additional Practice
  - Load a local Excel file and explore its structure
  - Create a DataFrame from a dictionary and add a new calculated column
  - Save filtered data to a new CSV file

Day

04

Data Cleaning and Preparation with Pandas

Handling Missing Values

- Why Handle Missing Values?
- Methods to Handle Missing Values
  - Drop Missing Values
  - Fill Missing Values
  - Interpolation

Data Transformations

- Renaming Columns
- Changing Data Types
- Creating or Modifying Columns

Combining and Merging DataFrames

- Concatenation
- Merging
- Joining

Hands-On Exercises

- Exercise 1: Clean a Dataset by Handling Missing Values and Renaming Columns
- Exercise 2: Merge Two Datasets and Perform Data Transformations
- Additional Practice
  - Drop columns with more than 50% missing values
  - Merge three datasets and analyze relationships between them
  - Convert categorical data to numerical using one-hot encoding

Day

05

Data Aggregation and Grouping in Pandas

Grouping Data by Categories

- Why Group Data?
- *groupby* in Pandas
  - Operations
    - Iterate over groups
    - Apply aggregation

Aggregation Functions

- Using *groupby*
- Using *pivot_table*
- Custom Aggregation
  - Apply custom functions using .agg()

Calculating Summary Statistics for Grouped Data

- Common Statistics
  - Mean
  - Max
  - Min
- Multi-Aggregation

Hands-On Exercises

- Exercise 1: Group Data by a Categorical Column
- Exercise 2: Calculate Summary Statistics for Grouped Data
- Additional Practice
  - Create a dataset of sales data and group it by region or product category
  - Use pivot_table to calculate total sales per region and per year
  - Create a custom aggregation function to calculate the variance for each group

Day

06

Data Visualization with Matplotlib and Seaborn

Introduction to Matplotlib for Plotting

- What is Matplotlib?
- Basic Syntax

Basic Plots

- Line Plot
- Bar Chart
- Histogram
- Scatter Plot

Customizing Plots

- Add titles, axis labels, and legends
- Adjust colors and styles

Introduction to Seaborn for Advanced Visualizations

- What is Seaborn?
- Common Seaborn Plots
  - Heatmap
  - Pairplot

Hands-On Exercises

- Exercise 1: Create Basic Plots with Matplotlib
- Exercise 2: Create a Heatmap with Seaborn
- Additional Practice
  - Create a histogram with multiple datasets overlaid
  - Use Seaborn to create a violin plot or box plot for visualizing distributions
  - Combine multiple plots in a single figure using Matplotlib's subplot

Day

07

Exploratory Data Analysis (EDA) Project

Applying Data Manipulation and Visualization for EDA

- What is EDA?
- Steps in EDA
  - Data Cleaning
  - Data Transformation
  - Aggregation and Filtering

Identifying Patterns, Trends, and Correlations

- Visual Tools for Insights:
  - Line plots for trends over time
  - Bar charts for categorical comparisons
  - Scatter plots for relationships
  - Heatmaps for correlation analysis
- Key Patterns to Look For:
  - Relationships between variables (correlations)
  - Distribution of variables (histograms, boxplots)
  - Outliers or anomalies

Summary Statistics, Visual Insights, and Hypothesis Generation

- Summary Statistics
- Hypothesis Generation

Hands-On Project: EDA on a Sample Dataset

- Task 1: Perform Data Cleaning, Aggregation, and Filtering
- Task 2: Generate Visualizations to Illustrate Key Insights
- Task 3: Identify and Interpret Patterns or Anomalies
- Task 4: Summarize Findings in a Report
- Additional Practice

    Use another dataset and apply the same EDA steps

    Explore advanced visualizations like boxplots or pairplots in Seaborn

    Create a dashboard for your findings using Plotly or Dash.

WEEK 3: Mathematics for Machine Learning

- Day 1 Linear Algebra Fundamentals
- Day 2 Advanced Linear Algebra Concepts
- Day 3 Calculus for Machine Learning - Derivatives
- Day 4 Calculus for Machine Learning - Integrals and Optimization
- Day 5 Probability Theory and Distributions
- Day 6 Statistics Fundamentals
- Day 7 Math-Driven Mini Project – Linear Regression from Scratch

Day

01

Linear Algebra Fundamentals

Vectors and Matrices

- Vector

  Example:[2,3,4]

- Matrix

$$\begin{bmatrix} 2 & -3 & 1 \\ 2 & 0 & -1 \\ 1 & 4 & 5 \end{bmatrix}$$

- Properties

Matrix Operations

- Addition and Subtraction
- Scalar Multiplication
- Matrix Multiplication

Formula: $(A \cdot B)_{ij} = \sum_k A_{ik} B_{kj}$

Special Matrices

- Identity Matrix (I)

  $I \cdot A = A$

- Zero Matrix (0)
- Diagonal Matrix

Hands-On Exercises

- Exercise 1: Create Vectors and Matrices Using NumPy
- Exercise 2: Implement Matrix-Vector Multiplication
- Exercise 3: Explore Special Matrices
- Additional Practice
  - Compute the determinant and inverse of a 2×2 matrix using NumPy
  - Verify properties of matrix multiplication
  - Create a block diagonal matrix using NumPy

Day

02

Advanced Linear Algebra Concepts

Determinants and Inverse of a Matrix

- Determinants
  - Scalar value that provides information about a matrix's properties
  - Only for square matrices
  - det(A) = 0, the matrix A is singular
  - det(A) != 0, A is invertible
  - Geometric Interpretation
    - For a 2×2 matrix, the determinant represents the scaling factor of the area formed by its column vectors
  - Formula for 2 × 2 Matrix: $$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

Determinants and Inverse of a Matrix

- Inverse of a Matrix
  - denoted as A^{-1}
  - Product of a matrix and its inverse is the identity matrix: $A \times A^{-1} = I$
  - Matrix is invertible only if $\det(A) \neq 0$
  - Formula for 2 × 2 2×2 Matrix

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Eigenvalues and Eigenvectors

- What are Eigenvalues and Eigenvectors?
- If $A \cdot v = \lambda \cdot v$, then
  - v is an eigenvector
  - λ is the eigenvalue
- Geometric Interpretation
  - Eigenvectors point in the direction where the matrix transformation stretches or compresses vectors
  - Eigenvalues indicate the factor of stretching or compression
- Properties
  - Matrix of size $n \times n$ has n n eigenvalues and eigenvectors
  - Eigenvalues can be real or complex
  - For a symmetric matrix, eigenvalues are always real.
- Computing Eigenvalues and Eigenvectors in NumPy

Introduction to Matrix Decomposition

- What is Matrix Decomposition?
  - Process of breaking a matrix into simpler components to analyze or solve problems
- Singular Value Decomposition (SVD)
  - SVD decomposes a matrix A A into three matrices: $A = U \cdot \Sigma \cdot V^T$
    - U: Left singular vectors (orthogonal matrix)
    - Σ: Diagonal matrix of singular values (non-negative).
    - V^T: Right singular vectors (orthogonal matrix).
- Applications of SVD
- Computing SVD in NumPy

Hands-On Exercises

- Exercise 1: Calculate Determinant and Inverse of a Matrix
- Exercise 2: Compute Eigenvalues and Eigenvectors
- Exercise 3: Perform Singular Value Decomposition
- Additional Practice
  - Compute eigenvalues and eigenvectors for larger matrices
  - Use SVD to reduce the dimensionality of a dataset
  - Verify the property of eigenvalues: $\det(A - \lambda I) = 0$

Day

03

Calculus for Machine Learning (Derivatives)

Introduction to Derivatives and Their Role in Optimization

- What are Derivatives?
  - Measures the rate at which a function changes with respect to its input
  - For a function $f(x)$, the derivative $f'(x)$ indicates the slope of the tangent line at a point x
- Role in Optimization
- Common Derivatives

For $f(x) = x^2$, $f'(x) = 2x$.

For $f(x) = \sin(x)$, $f'(x) = \cos(x)$.

Partial Derivatives

- Partial Derivatives
  - Measures how a function changes with respect to one variable while keeping other variables constant
  - For $f(x, y) = x^2 + y^2$, partial derivatives are:

$$\frac{\partial f}{\partial x} = 2x, \quad \frac{\partial f}{\partial y} = 2y$$

Gradients

- Gradient
  - Vector of all partial derivatives, indicating the direction of the steepest ascent
  - For $f(x, y) = x^2 + y^2$, the gradient is:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

Gradient Descent Optimization Algorithm

- What is Gradient Descent?
  - Iterative optimization algorithm used to minimize a function
  - Updates parameters in the direction of the negative gradient to find the minimum
- Update Rule: $\theta = \theta - \alpha \nabla f(\theta)$
  - θ: Parameters of the model
  - α: Learning rate (step size)
- Why is Gradient Descent Important in Machine Learning?

Hands-On Exercises

- Exercise 1: Compute Derivatives of Basic Functions
- Exercise 2: Compute Gradients
- Exercise 3: Implement Gradient Descent for Linear Regression
- Additional Practice
  - Use sympy to compute second-order derivatives (Hessian matrix)
  - Implement gradient descent with multiple learning rates and compare convergence speeds
  - Visualize the gradient descent process on a quadratic function.

Day

04

Calculus for Machine Learning (Integrals and Optimization)

Understanding Integrals and their Applications in ML

- What Are Integrals?
  - Compute the area under a curve, representing accumulation
  - The definite integral of $f(x)$ from a to b:

$$\int_a^b f(x)\,dx$$

- Applications in ML
  - Probability Distributions
  - Cost Functions

Optimization Concepts

- Local vs. Global Minima
  - Local Minimum
  - Global Minimum
- Convex Functions
  - $f(\lambda x1 + (1 - \lambda) x2) \leq \lambda f(x1) + (1 - \lambda) f(x2)$ for all $\lambda \in [0, 1]$
  - Ensure that any local minimum is also a global minimum
- Non-Convex Functions in ML
  - Most neural network loss functions

Stochastic Gradient Descent (SGD) and Its Variants

- What is Stochastic Gradient Descent?
  - Optimization algorithm that uses random subsets (mini-batches) of the data to compute gradients and update parameters
- Why Use SGD?
- Variants of SGD
  - Mini-Batch SGD
  - Momentum
  - Adam Optimizer

Hands-On Exercises

- Exercise 1: Calculate Integrals of Simple Functions
- Exercise 2: Implement Stochastic Gradient Descent for a Linear Model
- Additional Practice
  - Visualize the loss function's surface and the SGD optimization path
  - Implement Mini-Batch SGD and compare it with vanilla SGD
  - Use Adam optimizer for a more complex dataset

Day

05

Probability Theory and Distributions

Probability Basics

- Conditional Probability

  The probability of an event A given that event B has occurred

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

- Bayes' Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

  - P(A) Prior probability
  - P (B | A) Likelihood
  - P (B) Evidence

Common Probability Distributions

- Gaussian (Normal) Distribution
  - Bell-shaped curve with mean (μ) and standard deviation (σ)

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Bernoulli Distribution
  - Describes outcomes of a binary experiment

$$P(X = 1) = p, \quad P(X = 0) = 1 - p$$

Common Probability Distributions

- Binomial Distribution
  - Models the number of successes in n independent Bernoulli trials

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

- Poisson Distribution
  - Models the number of events in a fixed interval of time or space

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Applications in Machine Learning

- Gaussian Distribution
  - Used in Gaussian Naive Bayes and kernel density estimation
- Bernoulli Distribution
  - Models binary classification problems
- Binomial Distribution
  - Used in logistic regression to model binary outcomes.
- Poisson Distribution
  - Models count data

Hands-On Exercises

- Exercise 1: Calculate Probabilities Using Bayes' Theorem
- Exercise 2: Plot and Explore Different Probability Distributions
- Additional Practice
  - Create and visualize a multinomial distribution for multi-class data
  - Compare Gaussian and uniform distributions for continuous data
  - Use probability distributions to simulate and analyze real-world datasets

Day

06

Statistics Fundamentals

Measures of Central Tendency and Dispersion

- Central Tendency
  - Mean: The average value of a dataset
  - Median: The middle value when data is sorted
  - Mode: The most frequently occurring value
- Dispersion
  - Variance: The average squared deviation from the mean
  - Standard Deviation: The square root of variance, indicating the spread of data. python Copy code

Hypothesis Testing

- What is Hypothesis Testing?
- Steps
  - Formulate Hypotheses
    - Null Hypothesis
    - Alternative Hypothesis
  - Calculate Test Statistic
  - Determine P-Value
  - Interpret Results

Confidence Intervals and Statistical Significance

- Confidence Interval
  - Range of values within which the true population parameter is expected to lie

$$\mathrm{CI} = \bar{x} \pm z \cdot \frac{s}{\sqrt{n}}$$

x: Sample mean

z: Z-score

s: Standard deviation

- Statistical Significance

Hands-On Exercises

- Exercise 1: Calculate Mean, Variance, and Standard Deviation
- Exercise 2: Perform a T-Test
- Additional Practice
  - Visualize the distribution of data and highlight mean, median, and mode using Matplotlib
  - Perform hypothesis testing on real-world datasets (e.g., comparing exam scores of two groups)
  - Calculate confidence intervals for proportions in a dataset.

Day

07

Math-Driven Mini Project
Linear Regression from Scratch

Applying Linear Algebra, Calculus, and Statistics

- Linear Algebra
  - Mathematical Model: $\hat{y} = X \cdot \theta$

    X: Feature matrix (with bias term) | θ: Parameters (weights and bias) | y^: Predicted values.
- Calculus
  - Optimization of θ involves minimizing the loss function

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

Gradient of $J(\theta)$:

$$\nabla J = \frac{1}{m} X^T (X \cdot \theta - y)$$

- Statistics
  - Metrics like Mean Squared Error (MSE) and R^2 are used to evaluate model performance

Using Gradient Descent for Parameter Optimization

- Gradient Descent Algorithm
  - Iteratively update θ using $\theta := \theta - \alpha \cdot \nabla J$

    α: Learning rate
- Key Steps
  - Initialize parameters (θ)
  - Compute gradients (∇J)
  - Update parameters using the gradient descent rule

Evaluating the Model Using Statistical Metrics

- Mean Squared Error (MSE)

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

  - Measures the average squared error

- R-squared (R^2)

$$R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}$$

  - Measures how well the regression line explains the variance in the data

Hands-On Project: Linear Regression from Scratch

- Task 1: Implement the Mathematical Formula for Linear Regression
- Task 2: Use Gradient Descent to Optimize the Model Parameters
- Task 3: Calculate Evaluation Metrics

WEEK 4: Probability and Statistics for Machine Learning

Day 1 Probability Theory and Random Variables

Day 2 Probability Distributions in Machine Learning

Day 3 Statistical Inference - Estimation and Confidence Intervals

Day 4 Hypothesis Testing and P-Values

Day 5 Types of Hypothesis Tests

Day 6 Correlation and Regression Analysis

Day 7 Statistical Analysis Project – Analyzing Real-World Data

Day

01

Probability Theory and Random Variables

Basic Probability Concepts

- Sample Space and Events
  - Sample Space: The set of all possible outcomes of a random experiment
  - Events: A subset of the sample space
- Conditional Probability
  - The probability of an event A occurring, given that B has occurred
- Independence
  - Two events A and B are independent if
    - $P(A \cap B) = P(A) \cdot P(B)$
- Example with Python

Random Variables

- What are Random Variables?
  - Maps outcomes of a random experiment to numerical values
  - Types: Discrete | Continuous
- Probability Mass Function (PMF)
  - Probability distribution of a discrete random variable
- Probability Density Function (PDF)
  - Probability distribution of a continuous random variable

Expectation, Variance, and Standard Deviation

- Expectation ($E[X]$)

  Weighted average of a random variable's possible values

$$E[X] = \sum_{x} x \cdot P(X = x) \quad \text{(Discrete)}$$

$$E[X] = \int_{-\infty}^{\infty} x \cdot f(x)\, dx \quad \text{(Continuous)}$$

- Variance ($Var[X]$)

  Measures the spread of a random variable

$$Var[X] = E[(X - E[X])^2]$$

- Standard Deviation (σX)

  Square root of variance:

$$\sigma_X = \sqrt{Var[X]}$$

Hands-On Exercises

- Exercise 1: Simulate Dice Rolls and Calculate Probabilities
- Exercise 2: Create and Analyze Random Variables
- Additional Practice
  - Simulate flipping a coin 10,000 times and calculate probabilities of heads/tails
  - Compute the expectation and variance of a weighted die (biased probabilities)
  - Explore other distributions (e.g., normal, binomial) using Python

Day

02

Probability Distributions in Machine Learning

Common Probability Distributions

- Gaussian (Normal) Distribution
  - Bell-shaped curve characterized by mean (μ) and standard deviation (σ)
  - Probability Density Function (PDF)

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

  - Properties
    - Symmetric about the mean
    - Mean, median, and mode are the same
  - Application in ML
    - Common assumption in many algorithms (e.g., Naive Bayes)
    - Used in feature scaling (e.g., standardization)

Common Probability Distributions

- Binomial Distribution
  - Models the number of successes in n n independent Bernoulli trials
  - Probability Mass Function (PMF)

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

  - Properties
    - Discrete distribution
    - Parameters: n(number of trials), p(probability of success)
  - Application in ML
    - Logistic regression assumes a binomial distribution for binary classification

Common Probability Distributions

- Poisson Distribution
  - Models the number of events in a fixed interval
  - Probability Mass Function (PMF)

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

  - Properties
    - Discrete distribution
    - Parameter: λ (average rate of occurrence)
  - Application in ML
    - Used in event modeling

Common Probability Distributions

- Uniform Distribution
  - Equal probability for all outcomes in a range
  - Probability Density Function (PDF)

$$f(x) = \frac{1}{b-a}, \quad x \in [a, b]$$

  - Properties
    - Continuous distribution
    - Parameters: a(lower bound), b(upper bound)
  - Application in ML
    - Random initialization of weights in neural networks

Application of Distributions in Machine Learning

- Gaussian Distribution
  - Used in algorithms like Naive Bayes and Gaussian Mixture Models
  - Assumed in statistical tests
- Binomial Distribution
  - Foundational for logistic regression and other binary classification models
- Poisson Distribution
  - Applied in modeling count data
- Uniform Distribution
  - Commonly used in random sampling and initialization of parameters

Visualizing Distributions and Understanding Their Properties

- Visualization helps understand skewness, kurtosis, and outliers
- Skewness
  - Measure of symmetry: Positive skew | Negative skew
- Kurtosis
  - Measure of the "tailedness" of the distribution: High kurtosis | Low kurtosis

Hands-On Exercises

- Exercise 1: Plot and Explore Various Distributions
- Exercise 2: Analyze a Dataset's Distribution
- Additional Practice
  - Compare the effects of skewness and kurtosis on different datasets
  - Simulate random variables from custom distributions
  - Explore datasets with real-world applications of distributions

Day

03

Statistical Inference - Estimation and Confidence Intervals

Introduction to Statistical Inference

- What is Statistical Inference?
  - Process of making conclusions about a population based on sample data
- Population vs. Sample
- Goal
  - Estimate population parameters and assess the reliability of these estimates

Point Estimation and Interval Estimation

- Point Estimation
  - Single value estimate of a population parameter
- Interval Estimation
  - Provides a range of values within which the population parameter is likely to lie
  - Confidence Interval (CI):

$$\text{CI} = \bar{x} \pm z \cdot \frac{s}{\sqrt{n}}$$

$\bar{x}$: Sample mean

z: Z-score corresponding to the confidence level

s:Sample standard deviation

n: Sample size

Constructing Confidence Intervals

- For Means

  When the population standard deviation (σ) is unknown

$$\text{CI} = \bar{x} \pm t \cdot \frac{s}{\sqrt{n}}$$

  Use the t-distribution for small samples ($n < 30$)

- For Proportions

$$\text{CI} = \hat{p} \pm z \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

  p^: Sample proportion

Hands-On Exercises

- Exercise 1: Calculate Confidence Intervals for Sample Data
- Exercise 2: Conduct Sampling and Create a Report
- Additional Practice
  - Create confidence intervals for other statistics
  - Perform stratified sampling and compare intervals across strata
  - Visualize confidence intervals for multiple samples using Matplotlib

Day

04

Hypothesis Testing and P-Values

Introduction to Hypothesis Testing

- What is Hypothesis Testing?
  - Statistical method to determine if there is enough evidence in a sample to infer a conclusion about the population
- Key Components
  - Null Hypothesis: Assumes no effect or no difference
  - Alternative Hypothesis: Indicates an effect or difference
- Steps in Hypothesis Testing
  - Formulate Null Hypothesis and Alternative Hypothesis
  - Choose a significance level (α) — common values are 0.05 or 0.01
  - Calculate the test statistic
  - Determine the p-value
  - Compare the p-value to α:
    - If $p \leq \alpha$, reject Null Hypothesis | If $p > \alpha$, fail to reject Null Hypothesis

Understanding P-Values and Significance Levels

- P-Value
  - The probability of observing results as extreme as the test statistic under Null Hypothesis
  - Smaller p-values indicate stronger evidence against Null Hypothesis
- Significance Level (α)
  - Threshold for deciding whether to reject
  - Example: α = 0.05 means a 5% risk of rejecting Null Hypothesis when it is true
- Decision Rule
  - Reject Null Hypothesis: $p \leq \alpha$
  - Fail to Reject Null Hypothesis: $p > \alpha$

Types of Errors

- Type I Error (α)
  - Incorrectly rejecting Null Hypothesis when it is true
  - Example: Concluding a drug is effective when it is not
- Type II Error (β)
  - Failing to reject Null Hypothesis when it is false
  - Example: Concluding a drug is not effective when it is
- Example:

Reality	Decision	Conclusion
H_0 is true	Reject H_0	Type I
H_0 is true	Fail to reject H_0	Correct
H_0 is false	Reject H_0	Correct
H_0 is false	Fail to reject H_0	Type II

Hands-On Exercises

- Exercise 1: Perform a Hypothesis Test
- Exercise 2: Two-Sample T-Test
- Additional Practice
  - Perform a z-test for large sample sizes
  - Use the Iris dataset to test if the mean sepal length differs between two species
  - Perform hypothesis testing on proportions using the binomial distribution

Day

05

Types of Hypothesis Tests

T-Tests

- Purpose: Test whether the means of one or more groups differ significantly
- Types
  - One-Sample T-Test: Tests if the mean of a sample differs from a known value or population mean
  - Two-Sample T-Test (Independent T-Test): Compares the means of two independent groups
  - Paired Sample T-Test: Compares means of two related groups (e.g., pre-test vs. post-test)
- Example Use Cases
  - One-Sample: Testing if the average test score of a class differs from the national average
  - Two-Sample: Comparing test scores between two classes
  - Paired Sample: Comparing weight before and after a diet program

Chi-Square Test

- Purpose: Test for independence or goodness-of-fit in categorical data
- Chi-Square Test of Independence: Tests if two categorical variables are independent
- Example Use Case: Testing if gender is independent of preference for a product
- Steps
  - Create a contingency table
  - Calculate expected frequencies
  - Compute χ^2 statistic and p-value
- Python Implementation

ANOVA (Analysis of Variance)

- Purpose: Compare the means of three or more groups
- Hypotheses
    - Null: All group means are equal
    - Alternative: At least one group mean is different
- Example Use Case: Testing if the mean scores of students from three different schools differ
- Python Implementation

Hands-On Exercises

- Exercise 1: Conduct T-Tests
- Exercise 2: Perform a Chi-Square Test
- Exercise 3: Conduct ANOVA
- Additional Practice
  - Perform a two-way ANOVA to test for interaction effects
  - Use real-world datasets (e.g., student scores by gender and class) for hypothesis testing
  - Visualize test results using boxplots or bar plots.

Day

06

Correlation and Regression Analysis

Understanding Correlation

- What is Correlation?
  - Measures the strength and direction of the relationship between two variables
  - Values range from – 1 to 1, with 0 indicating no correlation
- Types of Correlation
  - Pearson Correlation Coefficient (r)
  - Spearman Correlation Coefficient (ρ)

Linear Regression Basics

- What is Linear Regression?
  - Method to model the relationship between a dependent variable (y) and one or more independent variables (x)
  - Formula for simple linear regression:

$$y = \beta_0 + \beta_1 x + \epsilon$$

    - β_0: Intercept
    - β_1: Slope
    - ϵ : Error term

Linear Regression Basics

- Key Metrics
  - Slope (β_1)
  - Intercept (β_0)
  - R-Squared (R^2)

Interpreting Regression Results

- Slope (β_1)
  - Indicates the magnitude and direction of the relationship
- Intercept (β_0)
  - Starting point of the regression line
- R-Squared (R^2)
  - Closer to 1 indicates better fit

Hands-On Exercises

- Exercise 1: Calculate Correlation Between Features
- Exercise 2: Fit a Simple Linear Regression Model
- Additional Practice
  - Fit a multiple linear regression model with multiple independent variables
  - Compare correlation and regression results for non-linear relationships
  - Use real-world datasets (e.g., housing prices) for regression analysis

Day

07

Statistical Analysis Project – Analyzing Real-World Data

Applying Probability and Statistical Concepts

- Previously learned statistical methods to draw meaningful insights from real-world data
- Apply
  - Probability concepts to understand distributions
  - Hypothesis testing to identify significant relationships or differences
  - Regression analysis to explore and quantify relationships between variables

Performing Exploratory Data Analysis (EDA)

- Steps in EDA
  1. Load and inspect the dataset
  2. Check for missing or inconsistent data
  3. Visualize distributions and relationships using histograms, scatter plots, and correlation heatmaps
- Key Goals
  - Understand the data structure
  - Identify patterns, trends, and outliers

Conducting Hypothesis Testing

- Steps
    1. Formulate null and alternative hypotheses
    2. Choose and perform an appropriate hypothesis test
    3. Interpret p-values and test results
- Example: Comparing Tip Amounts by Gender

Applying Linear Regression

- Steps
  1. Select dependent and independent variables
  2. Fit a regression model to the data
  3. Interpret coefficients and the R^2 value
- Example: Analyzing Relationship Between Total Bill and Tip

Hands-On Project

- Statistical Analysis of Real-World Data
  1. Perform Exploratory Data Analysis
  2. Conduct Hypothesis Tests
  3. Apply Linear Regression
- Additional Practice
  - Extend the project by exploring additional relationships (e.g., day of the week vs. tip amount).
  - Perform multiple linear regression with additional variables (e.g., include smoking status).
  - Use another real-world dataset (e.g., healthcare or sales data) to apply similar techniques.

WEEK 5: Introduction to Machine Learning

Day 1 Machine Learning Basics and Terminology

Day 2 Introduction to Supervised Learning and Regression Models

Day 3 Advanced Regression Models – Polynomial Regression and Regularization

Day 4 Introduction to Classification and Logistic Regression

Day 5 Model Evaluation and Cross-Validation

Day 6 k-Nearest Neighbors (k-NN) Algorithm

Day 7 Supervised Learning Mini Project

Day

01

Machine Learning Basics and Terminology

What is Machine Learning?

- Machine Learning
- Real-World Applications
  - Healthcare
  - Finance
  - E-commerce
  - Autonomous Vehicles
  - Natural Language Processing
- Why is ML Important?

Types of Machine Learning

- Supervised Learning
  - Model is trained on labeled data
  - Model learns to map inputs (features) to outputs (target)
  - Examples: Classification | Regression
  - Key Features
    - Requires labeled data
    - Accuracy depends heavily on the quality of the training data

Types of Machine Learning

- Unsupervised Learning
  - Model works on unlabeled data to find hidden patterns or structures
  - Examples: Clustering | Dimensionality Reduction
  - Key Features
    - No labeled data is needed
    - Focused on exploratory analysis and identifying patterns

Types of Machine Learning

- Reinforcement Learning
  - An agent interacts with an environment and learns by trial and error to maximize cumulative rewards
  - Examples: Robotics | Gaming | Dynamic Systems
  - Key Features
    - Goal-oriented learning based on rewards and penalties
    - Suitable for sequential decision-making problems

Key Concepts

- Features
  - The input variables (independent variables) used to train the model
  - Example: In predicting house prices, features could include the number of bedrooms, size, and location
- Target
  - The output variable (dependent variable) the model predicts
  - Example: House price is the target variable
- Training and Testing Datasets
  - The data is split into two subsets: Training Set | Testing Set
  - A typical split is 80% training and 20% testing

Key Concepts

- Overfitting
  - Model learns noise and details in the training data, performing poorly on new data
  - Model becomes too complex for the dataset
- Underfitting
  - Model is too simple to capture the underlying patterns in the data
  - Example: Fitting a linear model to non-linear data
- Bias-Variance Tradeoff
  - Bias: The error introduced by assuming a simplified model
  - Variance: Error introduced by the model's sensitivity to small changes in the training data
  - Goal: Balance bias and variance to achieve optimal performance.

Hands-On Exercise

- Define Features and Target Variables
- Split Data into Training and Testing Sets
- Visualize the Dataset

Day

02

Introduction to Supervised Learning and Regression Models

Overview of Supervised Learning

- Key Characteristics of Supervised Learning
  - Labeled Data
    - Supervised learning requires a dataset with labeled examples
    - Example: Inputs | Outputs
  - Objective
    - Minimize the error between the predicted output and the actual output
  - Types of Supervised Learning
    - Regression: Predicts continuous outputs
    - Classification: Predicts discrete outputs

Overview of Supervised Learning

- Applications of Supervised Learning
  - Healthcare
    - Predicting patient outcomes based on medical data
  - Finance
    - Fraud detection in transactions, credit risk assessment
  - Retail
    - Personalized product recommendations based on customer behavior
  - Autonomous Vehicles
    - Object detection and lane tracking using image classification

Introduction to Regression Analysis

- Linear Regression
  - Assumes a linear relationship between the dependent variable (y) and the independent variable (x)
  - Equation of a Line: $\mathbf{y = \beta_0 + \beta_1 x + \epsilon}$
    - $\boldsymbol{\beta_0}$: Intercept of the line
    - $\boldsymbol{\beta_1}$: Slope of the line
    - $\boldsymbol{\epsilon}$: Error term representing the difference between the observed and predicted values
  - Steps in Linear Regression
    - Fit the Model
    - Predict
    - Evaluate
  - Applications of Regression Models
    - Predicting Sales, Healthcare, Real Estate

Cost Function and Optimization in Linear Regression

Linear regression aims to minimize the error between the predicted and actual values of the target variable. This is achieved using a **cost function**

- Cost Function
  - Measures how far the predictions are from the actual values
  - Most common cost function is the Mean Squared Error (MSE)

$$J(\beta_0, \beta_1) = \frac{1}{m} \sum_{i=1}^{m} (y_i - (\beta_0 + \beta_1 x_i))^2$$

    - m: Number of data points
    - y_i : Actual value of the target for the i-th data point
    - $\beta_0 + \beta_1 x_i$: Predicted value for the i i-th data point
  - Objective
    - Minimize $J(\beta_0, \beta_1)$ to find the best-fitting line

Cost Function and Optimization in Linear Regression

- Optimization with Gradient Descent
  - Gradient Descent Algorithm
    - Iteratively updates β_0 and β_1 to minimize the cost function
    - Update Rule

$$\beta_j := \beta_j - \alpha \frac{\partial J}{\partial \beta_j}$$

      - α: Learning rate controlling the step size
  - Convergence
    - Algorithm stops when the updates become very small or a predefined number of iterations is reached
- Visualizing Optimization
  - The optimization process can be visualized as finding the lowest point on a cost surface

Hands-On Exercise

- Implement a Simple Linear Regression Model Using Scikit-Learn
- Visualize the Regression Line and Analyze Performance

Day

03

Advanced Regression Models

Polynomial Regression and Regularization

Polynomial Regression for Modeling Non-Linear Relationships

Polynomial regression is an extension of linear regression that models non-linear relationships by introducing higher-order terms of the input features.

- What is Polynomial Regression?
  - In a typical linear regression

    $$y = \beta_0 + \beta_1 x + \epsilon$$

  - In polynomial regression, we extend this to include higher-degree terms

    $$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n + \epsilon$$

  - The inclusion of higher-order terms allows the model to capture non-linear patterns in the data.

Polynomial Regression for Modeling Non-Linear Relationships

- Steps in Polynomial Regression
  1. Feature Transformation
     - Create polynomial features from the original input data
     - Example: $x \rightarrow [x, x^2, x^3]$
  2. Model Training
     - Perform linear regression on the transformed features
  3. Evaluation
     - Assess the model's ability to capture the data's non-linear structure
- Advantages
  - Captures non-linear relationships effectively
- Limitations
  - Prone to overfitting with high-degree polynomials
  - May require regularization to avoid overfitting
- Example Use Case: Predicting growth patterns in biological systems where relationships are non-linear

Introduction to Regularization Techniques: Lasso and Ridge Regression

- What is Regularization?
  - Technique used to prevent overfitting by adding a penalty term to the cost function of a regression model
- Types of Regularization
  - Ridge Regression (L2 Regularization)
    - Adds the sum of the squared coefficients to the cost function

$$J(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

    - λ parameter controls the strength of the penalty
  - Lasso Regression (L1 Regularization)
    - Adds the sum of the absolute coefficients to the cost function

$$J(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

    - Encourages sparsity in the coefficients, effectively performing feature selection
- Key Differences
  - Ridge shrinks coefficients but does not eliminate them
  - Lasso can shrink some coefficients to zero, removing irrelevant features.

Avoiding Overfitting with Regularization

- Regularization reduces the risk of overfitting by controlling the complexity of the model
- The regularization parameter λ (also called α in some libraries) plays a critical role:
  - A high λ value increases the penalty, forcing smaller coefficients and reducing overfitting
  - A low λ value allows the model to fit the training data more closely, increasing the risk of overfitting
- Example Code for Lasso and Ridge Regression

Hands-On Exercise

- Implement Polynomial Regression and Visualize the Fit
- Use Lasso and Ridge Regression
- Additional Experiments
  - Vary Regularization Parameters:
    - Experiment with different values of α (e.g., 0.1, 1, 10) for Ridge and Lasso regression
    - Observe how the model's coefficients and performance change
  - Use Multiple Features:
    - Include more features (e.g., HouseAge, AveRooms) and observe the impact on model performance
  - Feature Importance with Lasso:
    - Use Lasso regression to perform feature selection and identify the most relevant predictors

Day

04

Introduction to Classification and Logistic Regression

Classification Problems and Common Use Cases

- What is Classification?
- Types of Classification
  - Binary Classification
  - Multi-Class Classification
  - Multi-Label Classification
- Common Use Cases
  - Healthcare | Finance | Retail | Natural Language Processing | Autonomous Systems

Logistic Regression for Binary Classification

- What is Logistic Regression?
- The Logistic Regression Model
  - Equation
    - Logistic regression applies the sigmoid function to a linear equation

$$P(y = 1|x) = \sigma(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)$$

    - Where:
      - $P(y = 1 | x)$: Probability of the positive class
      - $\sigma(z)$: Sigmoid function

Logistic Regression for Binary Classification

- The Logistic Regression Model
  - Sigmoid Function
    - Maps the output to a range between 0 and 1

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

    - If $P(y = 1 \mid x) \geq 0.5$, the prediction is class 1; otherwise, it is class 0
  - Decision Boundary
    - The threshold (default is 0.5) used to classify instances
    - Decision boundaries can be adjusted to optimize for precision or recall
  - Interpretation of Coefficients:
    - β_0: Intercept, the baseline probability.
    - β_i: Effect of feature x_i on the log-odds of the positive class

Sigmoid Function, Decision Boundary, and Interpretation

- Sigmoid Function in Action
  - The sigmoid function ensures outputs are interpretable as probabilities
  - Visualization of the sigmoid function can illustrate how probabilities are mapped from raw model predictions
- Decision Boundary
  - Default threshold is 0.5
  - Adjusting the threshold can balance precision and recall depending on the use case

Hands-On Exercise

- Implement a logistic regression model to classify a dataset (e.g., predicting if a customer will make a purchase)
- Analyze model performance

Day

05

Model Evaluation and Cross-Validation

Model Evaluation Metrics for Regression and Classification

- Model Evaluation for Regression
  - Mean Squared Error (MSE)
    - Measures the average squared difference between predicted and actual values
    - Sensitive to outliers due to squaring of errors
  - Mean Absolute Error (MAE)
    - Measures the average absolute difference between predicted and actual values
    - Provides a more interpretable measure but less sensitive to outliers
  - Root Mean Squared Error (RMSE)
    - Square root of MSE, providing errors in the same units as the target variable

Model Evaluation Metrics for Regression and Classification

- Model Evaluation for Classification
  - Accuracy
    - Proportion of correctly predicted instances
    - Useful when the dataset is balanced
  - Precision
    - Fraction of positive predictions that are correct
    - Important for applications like fraud detection, where false positives are costly
  - Recall (Sensitivity)
    - Fraction of actual positives that are correctly identified
    - Useful in cases where missing positive instances is critical
  - F1 Score
    - Harmonic mean of precision and recall
    - Balances precision and recall, especially useful for imbalanced datasets

Introduction to Cross-Validation

- Key Cross-Validation Techniques
  - K-Fold Cross-Validation
    - Splits the dataset into K equal parts
    - Trains the model on K – 1 folds and tests on the remaining fold, repeating the process K times
    - The average of the K test scores provides the final evaluation metric
  - Stratified K-Fold
    - Ensures each fold has a proportional representation of classes in classification problems
  - Leave-One-Out Cross-Validation (LOOCV)
    - Trains the model on n – 1 samples and tests on the remaining one. Repeated for all samples
    - Computationally expensive for large datasets
- Advantages
  - Reduces the risk of overfitting by testing on multiple subsets of data
  - Provides a more generalized evaluation of model performance.

Understanding the Confusion Matrix

The confusion matrix is a table that summarizes the performance of a classification model by comparing predicted and actual values

- Structure of a Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

- Key Metrics Derived
  - True Positive Rate (TPR)
    - Same as recall
  - False Positive Rate (FPR)
    - Proportion of negatives incorrectly classified as positives
  - Specificity
    - Proportion of negatives correctly classified.

Hands-On Exercise

- Evaluate a model using cross-validation to obtain a more accurate estimate of model performance
- Generate and interpret a confusion matrix for a classification model

Day

06

k-Nearest Neighbors (k-NN) Algorithm

Introduction to k-Nearest Neighbors (k-NN) Algorithm and Its Applications

- What is k-Nearest Neighbors?
- Key Characteristics
  - Instance-Based Learning
  - Distance Metric
  - Classification
  - Regression
- Applications
  - Image Recognition
  - Recommendation Systems
  - Medical Diagnosis
  - Customer Segmentation

How k-NN Works for Classification and Regression

- Step-by-Step Process
  - Feature Scaling
  - Calculate Distances
  - Identify k Nearest Neighbors
  - Make Predictions
    - Classification
    - Regression

Choosing the Optimal Value of k

- Choosing k
  - Small k
    - High sensitivity to noise
    - Captures local variations in data
  - Large k
    - Smoother decision boundaries but can miss finer details
- Common Practices
  - Use cross-validation to determine the optimal value of k
  - A common starting point is $k = \sqrt{n}$, where n is the number of training samples

Understanding the Model's Limitations

- Computationally Expensive
  - Predictions require distance computation for all training samples
- Feature Scaling Dependence
  - Requires proper scaling to avoid feature dominance
- Not Robust to Imbalanced Data
  - Classes with more samples can dominate predictions

Hands-On Exercise

- Implement k-NN for a classification task, experimenting with different values of k
- Compare k-NN results to logistic regression, analyzing accuracy and other metrics

Day

07

Supervised Learning Mini Project

Building an End-to-End Supervised Learning Project

- Key Steps:
  - Define the Problem
    - Identify the objective: Regression or Classification
  - Data Preparation
    - Exploratory Data Analysis (EDA)
      - Understand the structure of the dataset
      - Visualize data distributions and relationships
    - Preprocessing
      - Handle missing values
      - Scale features for algorithms like k-NN
      - Encode categorical variables

Building an End-to-End Supervised Learning Project

- Key Steps:
  - Model Selection
    - Choose appropriate models based on the problem
  - Model Evaluation
    - Use performance metrics like Mean Squared Error (MSE) for regression or Accuracy, Precision, Recall, and F1 Score for classification
  - Comparison
    - Compare multiple models to identify the best-performing one

Applying Regression and Classification Models on a Real-World Dataset

- Dataset Options
  - Regression Example
    - Predict house prices using features like square footage, number of rooms, and location
    - Dataset: California Housing Dataset or any housing dataset
  - Classification Example
    - Classify customer churn based on customer demographics and behavior
    - Dataset: Telco Customer Churn Dataset

Evaluating and Comparing Model Performance

- Steps
  1. Evaluate models using cross-validation
  2. Generate performance metrics
  3. Summarize findings to identify strengths and weaknesses of each model

Hands-On Project

- Mini Project: Building a Supervised Learning Model
- Tasks
  - Task 1: Perform Exploratory Data Analysis and Preprocessing
  - Task 2: Train and Evaluate Multiple Models
  - Task 3: Summarize Findings in a Report

WEEK 6: Feature Engineering and Model Evaluation

Day 1 Introduction to Feature Engineering

Day 2 Data Scaling and Normalization

Day 3 Encoding Categorical Variables

Day 4 Feature Selection Techniques

Day 5 Creating and Transforming Features

Day 6 Model Evaluation Techniques

Day 7 Cross-Validation and Hyperparameter Tuning

Day

01

Introduction to Feature Engineering

Importance of Feature Engineering in Machine Learning

- What is Feature Engineering?
  - Process of transforming raw data into meaningful inputs for machine learning models
- Why is Feature Engineering Important?
  - Improves Model Accuracy
  - Reduces Model Complexity
  - Enables Model Interpretability
  - Handles Data Challenges
- Key Applications
  - Finance | Healthcare | E-Commerce

Types of Features: Categorical, Numerical, Ordinal

- Categorical Features
  - Represent discrete categories or labels
  - Encoding Techniques
    - One-Hot Encoding
    - Label Encoding
- Numerical Features
  - Represent continuous or discrete numbers
  - Preprocessing Techniques
    - Scaling
- Ordinal Features
  - Represent categorical data with a meaningful order
  - Encoding Techniques
    - Ordinal Encoding

Overview of Feature Engineering Techniques

- Scaling
  - Ensures all features contribute equally to the model
  - Techniques: Min-Max Scaling | Standardization
- Encoding
  - Converts categorical data into numerical format
  - Techniques: One-Hot Encoding | Label Encoding
- Transformation
  - Applies mathematical functions to modify features
  - Examples: Log Transformation | Polynomial Features
- Feature Selection
  - Reduces the number of input features to improve model performance
  - Techniques: Statistical Methods | Recursive Feature Elimination (RFE)

Hands-On Exercise

- Load a dataset and explore its features, identifying categorical and numerical features
- Plan which feature engineering techniques might be most suitable for the dataset

Day

02

Data Scaling and Normalization

Importance of Scaling and Normalization in Machine Learning

- What is Scaling and Normalization?
  - Preprocessing techniques used to transform numerical features to a common range or distribution
- Why is Scaling and Normalization Important?
  - Improves Algorithm Performance
  - Ensures Fair Comparisons
  - Stabilizes Training

Methods: Min-Max Scaling, Standardization (Z-Score Scaling)

- Min-Max Scaling
  - Transforms features to a specified range, typically [0, 1]
  - Ensures all feature values are within the same range
  - Use Cases: k-NN or neural networks
  - Limitations: Sensitive to outliers, as extreme values can distort the scale
- Standardization (Z-Score Scaling)
  - Centers the data around zero and scales it to have a standard deviation of 1
  - Ensures a standard normal distribution for each feature
  - Use Cases: SVM, logistic regression, and PCA
  - Advantages: Handles outliers better than Min-Max scaling

When to Use Scaling and Normalization for Different Algorithms

- Algorithms That Require Scaling
  - Distance-Based Algorithms
    - k-NN, SVM, K-Means clustering
  - Gradient-Based Models
    - Linear regression, logistic regression, and neural networks
- Algorithms Less Sensitive to Scaling
  - Tree-Based Models
    - Decision Trees, Random Forests, Gradient Boosting

Hands-On Exercise

- Apply Min-Max scaling and standardization to a dataset using scikit-learn
- Observe the effects of scaling on model performance by training a k-NN classifier before and after scaling

Day

03

Encoding Categorical Variables

One-Hot Encoding, Label Encoding

- What Are Categorical Variables?
  - Binary Categorical Features: Gender (Male/Female)
  - Multi-Class Categorical Features: Country (USA, Canada, UK).
- One-Hot Encoding
  - Creates binary columns for each category in a categorical feature
  - Each row is marked with a 1 for its respective category and 0 elsewhere
  - Example: Feature: Color = ['Red', 'Blue', 'Green']

```
Red | Blue | Green
1   | 0    | 0
0   | 1    | 0
0   | 0    | 1
```

  - Applications
    - Categorical features with a small number of unique categories
    - Tree-based models, logistic regression, and neural networks

One-Hot Encoding, Label Encoding

- Label Encoding
  - Label Encoding assigns a unique integer to each category
  - Example: Red = 0, Blue = 1, Green = 2.
  - Applications:
    - Ordinal features where the order matters
    - Can introduce unintended ordinal relationships for nominal features
  - Limitations
    - Can mislead algorithms into interpreting categories as ordered, especially when the variable is nominal

Dealing with High-Cardinality Categorical Features

- High-cardinality categorical features contain a large number of unique categories
- Challenges
  - Dimensionality
  - Sparse Representation
- Solutions
  - Frequency Encoding
    - Replace categories with their occurrence frequency in the dataset
    - Example: City = ['NY', 'LA', 'NY', 'SF', 'LA'] Encoded: NY = 2, LA = 2, SF = 1.
  - Target Encoding
    - Replace categories with the mean of the target variable for each category

When to Use Different Encoding Techniques

Encoding Technique	Use Case
One-Hot Encoding	Nominal features with a small number of unique categories
Label Encoding	Ordinal features or when used with algorithms like tree-based models
Frequency Encoding	High-cardinality features in both regression and classification tasks
Target Encoding	High-cardinality features in supervised learning tasks

Hands-On Exercise

- Apply One-Hot Encoding and Label Encoding to a dataset with categorical variables
- Experiment with different encoding techniques and observe their impact on model performance

Day

04

Feature Selection Techniques

Introduction to Feature Selection

- What is Feature Selection?
  - Process of identifying and retaining the most relevant features (input variables) in a dataset while discarding irrelevant or redundant ones
- Why is Feature Selection Important?
  - Improves Model Performance
  - Reduces Overfitting
  - Enhances Interpretability
  - Increases Computational Efficiency
- When to Use Feature Selection?
  - High-Dimensional Data
  - Correlated Features
  - Reducing Complexity

Techniques for Feature Selection

- Filter Methods
  - Evaluate the relevance of features by analyzing their statistical properties in relation to the target variable
  - Examples: Correlation | Mutual Information
  - When to Use: Quick evaluation of features before training a model
- Wrapper Methods
  - Iteratively selects features by training and evaluating a model
  - Examples: Forward Selection | Backward Elimination
  - When to Use: Useful when feature interactions are important but computationally expensive
- Embedded Methods
  - Perform feature selection as part of the model training process
  - Examples: Lasso Regression | Tree-Based Models
  - When to Use: Effective when training tree-based models or regularized regression

Hands-On Exercise

- Use correlation and mutual information to select important features from a dataset
- Apply a tree-based model (e.g., Random Forest) to identify the most important features

Day

05

Creating and Transforming Features

Feature Creation

- What is Feature Creation?
  - Feature creation involves deriving new, meaningful features from existing ones to enhance a model's ability to capture important patterns in the data
- Examples of Feature Creation
  - Date-Time Features
  - Interaction Features
  - Aggregations
- Importance
  - Adds domain knowledge to the dataset
  - Captures hidden patterns and trends not evident in the original features

Transforming Features

- What is Feature Transformation?
  - Feature transformation modifies existing features to better suit the learning algorithm
- Common Transformations
  - Logarithmic Transformation
    - Reduces skewness in highly skewed distributions
  - Square Root Transformation
    - Moderately reduces skewness, often used for count data
  - Polynomial Transformation
    - Adds higher-order terms (x^2, x^3) to capture non-linear relationships
- Importance
  - Enhances the model's ability to fit non-linear relationships
  - Makes distributions more normal-like, aiding algorithms that assume normality

Importance of Feature Transformations in Non-Linear Relationships

- Transformations allow linear models to handle non-linear relationships
  - For example:
    - Polynomial transformations enable linear regression to model quadratic patterns
    - Logarithmic transformations stabilize variance and handle skewness

- By transforming features, models become more robust and capable of capturing complex patterns in data.

Hands-On Exercise

- Create new features from a date column (e.g., day of the week, month, year)
- Apply polynomial transformations to a dataset and compare model performance before and after transformation

Day

06

Model Evaluation Techniques

Evaluation Metrics for Regression

Regression Metrics

- Mean Absolute Error (MAE)
  - Measures the average magnitude of errors without considering their direction
  - Use Case: Suitable when all errors have equal importance
- Mean Squared Error (MSE)
  - Measures the average of squared differences between actual and predicted values
  - Use Case: Penalizes larger errors more than MAE, making it sensitive to outliers
- Root Mean Squared Error (RMSE)
  - Square root of MSE, providing errors in the same units as the target variable
  - Use Case: A common metric for interpretability in real-world units
- R-squared (R^2)
  - Measures how well the model explains the variability of the target variable
  - Use Case: Indicates the proportion of variance explained by the model

Evaluation Metrics for Classification

Classification Metrics

- Accuracy
  - Percentage of correctly classified instances
  - Use Case: Suitable for balanced datasets but misleading for imbalanced data
- Precision
  - Fraction of true positive predictions among all positive predictions
  - Use Case: Important when false positives are costly (e.g., spam detection)
- Recall (Sensitivity)
  - Fraction of true positives identified among all actual positives
  - Use Case: Critical in situations where missing positives is costly (e.g., medical diagnosis)
- F1 Score
  - Harmonic mean of precision and recall
  - Use Case: Useful for imbalanced datasets
- ROC-AUC
  - Measures the ability of the model to distinguish between classes
  - Use Case: Important for evaluating binary classifiers

Understanding When to Use Each Metric

- Regression
  - Use MAE for interpretability and uniform importance of errors
  - Use MSE/RMSE when larger errors need greater penalization
  - Use R^2 to explain variance but not as a sole performance metric
- Classification
  - Use accuracy for balanced datasets
  - Use precision and recall for imbalanced datasets, depending on the problem's focus (e.g., minimizing false positives or false negatives)
  - Use F1 score for a balanced evaluation of precision and recall
  - Use ROC-AUC for overall model performance evaluation in binary classification

Hands-On Exercise

- Exercise 1: Classification Model Evaluation
  - Objective
    - Train a classification model, calculate confusion matrix, and interpret precision, recall, and F1 score
- Exercise 2: Regression Model Evaluation
  - Objective
    - Train a regression model and evaluate its performance using MAE, MSE, and R^2

Day

07

Cross-Validation and Hyperparameter Tuning

Introduction to Cross-Validation

- What is Cross-Validation?
  - Technique used to assess how well a machine learning model generalizes to an independent dataset
- Types of Cross-Validation
  - K-Fold Cross-Validation
    - Splits the dataset into K folds of approximately equal size
    - The model is trained on K – 1 folds and validated on the remaining fold
    - This process is repeated K times, and the average performance is computed
  - Stratified K-Fold
    - Ensures that each fold maintains the same class distribution as the original dataset
    - Useful for imbalanced datasets
  - Leave-One-Out Cross-Validation (LOOCV)
    - Uses a single data point for validation and the rest for training
    - Repeats this process for all data points
    - Computationally expensive but provides the most robust evaluation

Hyperparameter Tuning

- What is Hyperparameter Tuning?
  - Hyperparameters are parameters that are not learned by the model but are set before training, tuning these hyperparameters is crucial for optimizing model performance
- Techniques for Hyperparameter Tuning
  - Grid Search
    - Exhaustively searches over a predefined hyperparameter space
    - Example: Testing all combinations of values for max_depth and learning_rate
  - Random Search
    - Randomly samples combinations of hyperparameters from the predefined space
    - More efficient than Grid Search when the parameter space is large
  - Importance of Hyperparameter Tuning
    - Prevents overfitting and underfitting by selecting the best configuration
    - Enhances model performance by optimizing critical settings.

Importance of Tuning Hyperparameters for Model Performance

- Without tuning, the model might not reach its optimal performance, leading to:
  - Underfitting: Model fails to capture the underlying patterns.
  - Overfitting: Model memorizes the training data and performs poorly on unseen data

Hands-On Project:
Feature Engineering and Model Evaluation

- Objective
  - Perform end-to-end feature engineering, model evaluation, and hyperparameter tuning on a dataset
- Tasks
  - Task 1: Perform Feature Engineering
  - Task 2: Train and Evaluate Models
  - Task 3: Apply Grid Search for Hyperparameter Tuning

WEEK 7: Advanced Machine Learning Algorithms

Day 1 Introduction to Ensemble Learning

Day 2 Bagging and Random Forests

Day 3 Boosting and Gradient Boosting

Day 4 Introduction to XGBoost

Day 5 LightGBM and CatBoost

Day 6 Handling Imbalanced Data

Day 7 Ensemble Learning Project – Comparing Models on a Real Dataset

Day

01

Introduction to Ensemble Learning

Concept of Ensemble Learning

- What is Ensemble Learning?
  - Machine learning technique that combines the predictions of multiple models to produce a final output
- Why Does Ensemble Learning Improve Performance?
  - Reduces Variance
  - Reduces Bias
  - Improves Robustness
- Applications
  - Fraud detection, medical diagnoses, recommendation systems, and predictive analytics

Types of Ensemble Methods

- Bagging (Bootstrap Aggregating)
  - Trains multiple models independently on different subsets of data created through bootstrapping
  - Combines predictions by averaging (regression) or majority voting (classification)
  - Example: Random Forest
  - Strengths: Reduces variance without increasing bias
- Boosting
  - Trains models sequentially, where each model focuses on correcting the errors made by the previous ones
  - Combines predictions through weighted averaging or voting
  - Examples: AdaBoost, Gradient Boosting, XGBoost, LightGBM
  - Strengths: Reduces both bias and variance by focusing on hard-to-predict instances

Types of Ensemble Methods

- Stacking
  - Combines predictions from multiple base models (of different types) using a meta-model to learn how to best combine their outputs
  - Strengths: Can utilize diverse model types to maximize performance

Overview of Commonly Used Ensemble Methods

- Random Forest
  - Combines multiple decision trees using bagging
  - Reduces overfitting common in individual decision trees
- Gradient Boosting
  - Sequentially builds models that minimize errors in the previous ones
  - Suitable for both regression and classification tasks
- AdaBoost
  - Adjusts model weights based on performance
  - Focuses on misclassified instances
- XGBoost
  - Optimized version of gradient boosting, known for speed and accuracy
- Voting Classifier
  - Combines predictions of multiple models using majority voting or averaging

Hands-On Exercise

- Build a basic ensemble model combining predictions from Linear Regression, Decision Tree, and k-NN to observe the impact on accuracy

Day

02

Bagging and Random Forests

Understanding Bagging (Bootstrap Aggregating)

- What is Bagging?
  - Ensemble learning technique that trains multiple models on different subsets of the data, created by random sampling with replacement
  - Regression: Average the predictions of individual models
  - Classification: Use majority voting to determine the final class
- Why Use Bagging?
  - Reduces Variance
  - Improves Robustness
- Applications
  - Bagging is commonly used with decision trees, which are prone to high variance

Introduction to Random Forests

- What is a Random Forest?
  - Ensemble learning method that builds multiple decision trees using bagging
  - Key Features of Random Forests
    - Bootstrap Sampling
    - Feature Randomness
    - Prediction Aggregation
  - Advantages
    - Handles both regression and classification tasks effectively
    - Works well with high-dimensional data
    - Reduces overfitting compared to single decision trees

Key Parameters in Random Forests

- Number of Trees (n_estimators)
  - The number of decision trees in the forest
  - Larger values reduce variance but increase computational cost
- Maximum Depth (max_depth)
  - Limits the depth of each tree to prevent overfitting
  - Shallower trees generalize better but may underfit
- Feature Selection (max_features)
  - Number of features to consider when looking for the best split
  - Options
    - sqrt | log2 | None
- Minimum Samples per Leaf (min_samples_leaf)
  - Minimum number of samples required in a leaf node
  - Prevents overly complex trees by ensuring each leaf contains enough samples.

Hands-On Exercise

- Train a Random Forest classifier on a dataset, tune its parameters, and evaluate its performance

Day

03

Boosting and Gradient Boosting

Concept of Boosting

- What is Boosting?
  - Ensemble technique that sequentially combines weak learners to form a strong learner
  - Each subsequent model focuses on correcting the errors made by previous models
- How Does Boosting Differ from Bagging?

Feature	Bagging	Boosting
Approach	Trains models independently on bootstrapped subsets.	Trains models sequentially to correct errors.
Purpose	Reduces variance by averaging predictions.	Reduces bias by focusing on difficult cases.
Examples	Random Forest	Gradient Boosting, AdaBoost

Gradient Boosting

- What is Gradient Boosting?
  - Boosting algorithm that builds models sequentially by minimizing a loss function using gradient descent
  - Iteratively adds weak learners to improve overall model performance
- How Gradient Boosting Works
  - Initialize Model: Start with a simple model, often predicting the mean of the target variable
  - Compute Residuals: Calculate the difference between the actual and predicted values
  - Fit Weak Learner: Train a weak model to predict the residuals
  - Update Prediction: Add the predictions of the weak learner to the overall model
  - Repeat: Continue adding weak learners until the desired number of iterations or a stopping criterion is reached

Gradient Boosting

- Key Parameters in Gradient Boosting
  - Learning Rate
    - Determines the contribution of each weak learner
    - Smaller values reduce overfitting but require more iterations
  - Number of Estimators
    - The number of weak learners (trees) added sequentially
    - Larger values improve learning but increase computation time
  - Regularization
    - Techniques like limiting tree depth or adding penalties to prevent overfitting

Understanding the Key Parameters

- Learning Rate (learning_rate)
  - Lower values improve model performance by reducing overfitting but require more iterations
  - Typical range: 0.01 to 0.3
- Number of Estimators (n_estimators)
  - Represents the number of trees added to the ensemble
  - Larger values can improve performance but risk overfitting
- Tree Depth (max_depth)
  - Limits the complexity of individual trees
  - Shallower trees generalize better but might underfit.

Hands-On Exercise

- Train and evaluate a Gradient Boosting model on a dataset, tune key parameters, and compare its performance with a Random Forest model

Day

04

Introduction to XGBoost

Overview of XGBoost

- What is XGBoost?
  - Advanced implementation of the Gradient Boosting algorithm designed for speed and performance
  - It introduces various enhancements that make it faster, more efficient, and capable of handling complex datasets
- Improvements Over Traditional Gradient Boosting
  - Speed
  - Handling Missing Data
  - Regularization
  - Custom Loss Functions
  - Tree Pruning

Key Features of XGBoost

- Handling Missing Data
  - Automatically assigns missing values to the branch that minimizes the loss function
  - Reduces preprocessing steps for datasets with missing values
- Regularization
  - Includes penalties for overly complex models, reducing overfitting
  - Hyperparameters
    - lambda: L2 regularization term
    - alpha: L1 regularization term
- Parallel Processing
  - Splits calculations for tree construction across multiple cores, significantly improving training time

Hyperparameters in XGBoost and How to Tune Them

- Key Hyperparameters
  - Learning Rate (eta)
    - Controls the contribution of each tree to the model
    - Typical range: 0.01–0.3
  - Number of Trees (n_estimators)
    - Determines the number of boosting rounds
    - Larger values may improve performance but increase computation time
  - Tree Depth (max_depth)
    - Limits the depth of trees, balancing bias and variance
    - Shallower trees generalize better, while deeper trees may overfit
  - Subsample
    - Fraction of data used to train each tree
    - Helps reduce overfitting; typical range: 0.5–1.0
  - Colsample_bytree
    - Fraction of features used for each tree split
    - Typical range: 0.5–1.0
  - Regularization Parameters: lambda and alpha control L2 and L1 regularization, respectively

Hands-On Exercise

- Train an XGBoost model on a dataset, tune hyperparameters using cross-validation, and compare its performance with a Gradient Boosting model

Day

05

LightGBM and CatBoost

Introduction to LightGBM

- What is LightGBM?
  - Implementation of Gradient Boosting designed to handle large datasets and high-dimensional data with speed and accuracy
  - Key Features of LightGBM:
    - Histogram-Based Splitting
    - Leaf-Wise Tree Growth
    - Support for GPU Training
    - Handling Sparse Data
  - Advantages
    - Faster training than XGBoost
    - Handles large datasets effectively
    - Reduces memory usage with histogram-based splitting
  - When to Use LightGBM
    - Large datasets with numerical features
    - Time-sensitive tasks requiring fast training

Overview of CatBoost

- What is CatBoost?
  - Gradient Boosting library developed specifically to handle categorical features without the need for preprocessing like one-hot encoding
  - Key Features of CatBoost:
    - Native Support for Categorical Data
    - Ordered Boosting
    - Robust to Overfitting
  - Advantages
    - Eliminates the need for manual encoding of categorical data
    - Reduces overfitting with robust boosting techniques
    - Easy to implement for datasets with many categorical features
  - When to Use CatBoost
    - Datasets with a high proportion of categorical features
    - Applications where overfitting is a concern

XGBoost, LightGBM, and CatBoost

Feature	XGBoost	LightGBM	CatBoost
Speed	Moderate	Fast	Fast
Handling Categorical Data	Requires encoding	Requires encoding	Native support
Memory Usage	Moderate	Low	Moderate
Tuning Complexity	Moderate	High (leaf-wise growth)	Low
Best Use Cases	General-purpose models	Large datasets	Categorical-heavy datasets

Hands-On Exercise

- Train and compare LightGBM, CatBoost, and XGBoost models on a dataset, focusing on their ability to handle large datasets and categorical data

Day

06

Handling Imbalanced Data

Problems Caused by Imbalanced Data in Classification Tasks

- What is Imbalanced Data?
  - Refers to datasets where one class significantly outnumbers the other(s)
  - Challenges with Imbalanced Data
    - Bias Toward Majority Class:
      - Machine learning models tend to prioritize the majority class due to its frequency
    - Misleading Evaluation Metrics
      - Metrics like accuracy can be misleading, as they do not account for class imbalance
    - Limited Information for Minority Class
      - Insufficient samples for the minority class can lead to underfitting
  - Applications with Imbalanced Data
    - Fraud detection, medical diagnosis, and anomaly detection

Techniques to Handle Imbalanced Data

- Resampling Techniques
  - Oversampling
    - Increase the number of minority class samples by duplicating or synthesizing new samples
    - Example: SMOTE (Synthetic Minority Over-sampling Technique), which generates synthetic examples
  - Undersampling
    - Reduce the number of majority class samples to balance the dataset
    - Risk: Loss of valuable information from majority class.

Techniques to Handle Imbalanced Data

- Algorithmic Solutions
  - Class Weights
    - Assign higher weights to the minority class during model training
    - Many algorithms (e.g., Logistic Regression, Random Forest) have built-in support for class weights
  - Anomaly Detection Models
    - Treat the minority class as anomalies, focusing the model on detecting them.

Techniques to Handle Imbalanced Data

- Evaluation Metrics for Imbalanced Data
  - F1-Score
    - Harmonic mean of precision and recall, focusing on both false positives and false negatives
  - ROC-AUC
    - Measures the ability to distinguish between classes across various threshold values
  - Precision-Recall Curve
    - Focuses on performance for the positive class

Hands-On Exercise

- Apply SMOTE to handle class imbalance, train a classifier, and evaluate its performance using metrics like ROC-AUC and F1-score

Day

07

Ensemble Learning Project – Comparing Models on a Real Dataset

Building and Evaluating Multiple Ensemble Models

- Why Compare Ensemble Models?
  - Excel in different scenarios
  - Helps identify the most effective model for a specific dataset or problem
- Ensemble Methods to Consider:
  - Bagging (e.g., Random Forest)
    - Reduces variance by averaging predictions from multiple independent models
    - Works well with high-variance models like decision trees.
  - Boosting (e.g., Gradient Boosting, XGBoost, LightGBM)
    - Reduces bias by sequentially correcting errors from previous models
    - Effective for complex patterns and imbalanced datasets.

Comparing Bagging and Boosting

- Bagging
  - Builds models independently using random subsets of data
  - Robust against overfitting with strong base learners
- Boosting
  - Sequentially builds models, focusing on hard-to-predict samples
  - Requires careful tuning to prevent overfitting.

Model Performance on Balanced vs. Imbalanced Data

- Challenges with Imbalanced Data
  - Models may prioritize the majority class, leading to poor performance on the minority class
- Evaluation Metrics
  - Accuracy
    - May not reflect true performance for imbalanced datasets
  - F1-Score
    - Balances precision and recall, focusing on the minority class
  - ROC-AUC
    - Evaluates the model's ability to distinguish between classes across thresholds

Hands-On Project

- Ensemble Learning and Model Comparison
  - Train and compare multiple ensemble models on a real-world dataset, analyzing their performance under balanced and imbalanced conditions

WEEK 8: Model Tuning and Optimization

Day 1 Introduction to Hyperparameter Tuning

Day 2 Grid Search and Random Search

Day 3 Advanced Hyperparameter Tuning with Bayesian Optimization

Day 4 Regularization Techniques for Model Optimization

Day 5 Cross-Validation and Model Evaluation Techniques

Day 6 Automated Hyperparameter Tuning with GridSearchCV and RandomizedSearchCV

Day 7 Optimization Project – Building and Tuning a Final Model

Day

01

Introduction to Hyperparameter Tuning

Parameters and Hyperparameters

- What are Parameters?
  - Values learned by a machine learning model during training
  - Adjusted to minimize the loss function and optimize predictions
  - Examples
    - Coefficients in Linear Regression
    - Weights and biases in Neural Networks
- What are Hyperparameters?
  - Settings defined before training that influence how the model learns from data
  - Not learned from the data but instead control the training process
  - Examples
    - Tree Depth
    - Learning Rate
    - Number of Estimators

Importance of Tuning Hyperparameters

- Why Tune Hyperparameters?
  - Improve Model Performance
    - Optimal hyperparameters help models generalize better, reducing overfitting and underfitting
  - Enhance Efficiency
    - Proper tuning can reduce training time and computational resources
  - Adapt to Problem-Specific Needs
    - Tailoring hyperparameters ensures the model fits the dataset's characteristics

Common Hyperparameters in Popular Models

- Decision Trees and Random Forests
  - Max Depth: Limits the depth of trees to avoid overfitting
  - Min Samples Split: Minimum samples required to split an internal node
  - Number of Estimators: Total number of trees in Random Forest
- Gradient Boosting Models
  - Learning Rate: Determines the contribution of each tree
  - Subsample: Fraction of training data used to train each tree
  - Max Depth: Limits the complexity of individual trees
- Neural Networks
  - Learning Rate: Step size for weight updates
  - Number of Layers: Determines the depth of the network
  - Batch Size: Number of samples per gradient update.

Hands-On Exercise

- Objective

  Train a model with default hyperparameters, evaluate its performance, and manually adjust a few hyperparameters to observe their impact on results

Day

02

Grid Search and Random Search

Introduction to Grid Search and Random Search

- What is Grid Search?
  - Method of hyperparameter tuning that systematically evaluates all possible combinations of hyperparameter values within a specified grid
  - How It Works:
    - Define a range of values for each hyperparameter
    - Train and evaluate the model on each combination of hyperparameter values
    - Select the combination that yields the best performance
- What is Random Search?
  - Alternative method where hyperparameter combinations are sampled randomly from the specified ranges
  - How It Works
    - Define ranges or distributions for each hyperparameter
    - Randomly sample a specified number of combinations
    - Train and evaluate the model for each sampled combination

Pros and Cons of Each Method

Feature	Grid Search	Random Search
Exhaustiveness	Evaluates all combinations	Randomly samples combinations
Time Efficiency	Computationally expensive for large grids	Faster for large parameter spaces
Exploration	Limited to predefined grid	Explores more diverse ranges
Best Use Case	Small parameter spaces	Large parameter spaces with time constraints

Practical Guidance on Choosing Search Ranges

- Start with Broad Ranges
  - Use Random Search to explore large parameter spaces and identify promising ranges
- Refine with Grid Search
  - Narrow the search space based on Random Search results and perform exhaustive Grid Search for fine-tuning
- Understand Model Sensitivity
  - Some hyperparameters (e.g., learning rate) require fine granularity, while others (e.g., number of trees) can have coarser steps.

Hands-On Exercise

- Objective

  Implement both Grid Search and Random Search for hyperparameter tuning, compare their efficiency, and analyze the impact on model performance

Day

03

Advanced Hyperparameter Tuning with Bayesian Optimization

Introduction to Bayesian Optimization

- What is Bayesian Optimization?
  - Advanced method for hyperparameter tuning that balances exploration (searching new regions) and exploitation (refining promising regions)
  - Uses a probabilistic model to guide the search for optimal hyperparameters
  - How It Works
    - Surrogate Model
      - Builds a probabilistic model (e.g., Gaussian Process) of the objective function based on prior evaluations
    - Acquisition Function
      - Balances exploration and exploitation by choosing the next hyperparameters to evaluate based on predicted performance and uncertainty
    - Iterative Refinement
      - Updates the surrogate model after each evaluation, refining the search
  - Why Use Bayesian Optimization?
    - Efficient for high-dimensional and expensive-to-evaluate functions
    - Reduces the number of evaluations required to find near-optimal hyperparameters

Using Libraries for Bayesian Optimization

- Popular Libraries
  - Hyperopt
    - Simplifies Bayesian Optimization for hyperparameter tuning
    - Works with *fmin* to minimize objective functions over a parameter space
  - Optuna
    - Flexible and user-friendly library for hyperparameter optimization
    - Supports dynamic search spaces and pruning of unpromising trials

Understanding Exploration vs. Exploitation

- Exploration
  - Focuses on sampling hyperparameters from unexplored regions
  - Useful for identifying new areas of high potential
- Exploitation
  - Focuses on refining the search around regions with known high performance
  - Useful for fine-tuning near-optimal hyperparameters
- Bayesian Optimization's Advantage
  - Balances these approaches using the acquisition function to minimize unnecessary evaluations while improving results.

Hands-On Exercise

- Objective

  Apply Bayesian Optimization using Optuna to tune an XGBoost model and compare the results with Grid Search and Random Search

Day

04

Regularization Techniques for Model Optimization

Understanding Overfitting and Underfitting

- Overfitting
  - Occurs when a model learns the noise in the training data along with the patterns, leading to poor generalization on unseen data
  - Symptoms
    - High training accuracy but low test accuracy
    - Large differences between training and validation losses
- Underfitting
  - Occurs when a model is too simple to capture the underlying patterns in the data
  - Symptoms
    - Low accuracy on both training and test sets
    - High bias in predictions

Regularization Techniques

- Regularization introduces a penalty term to the loss function during model training to prevent overfitting by discouraging overly complex models
  - L1 Regularization (Lasso)
    - Adds the absolute values of coefficients to the loss function
    - Encourages sparsity by setting some coefficients to zero, effectively selecting features.
  - L2 Regularization (Ridge)
    - Adds the squared values of coefficients to the loss function
    - Shrinks coefficients toward zero but does not set them to zero.
  - Elastic Net
    - Combines L1 and L2 regularization
    - Useful when there are correlated predictors and when feature selection is desired

Practical Applications of Regularization

- Prevent Overfitting
  - Penalizes large coefficients, reducing model complexity
- Handle Multicollinearity
  - Ridge regularization is effective when predictors are highly correlated
- Feature Selection
  - Lasso automatically performs feature selection by setting some coefficients to zero

Hands-On Exercise

- Objective
  - Apply Lasso and Ridge regularization on a linear regression model, compare performance, and analyze the effects on coefficients

Day

05

Cross-Validation and Model Evaluation Techniques

Importance of Cross-Validation in Model Evaluation

- What is Cross-Validation?
  - Statistical method used to evaluate the performance of a model by partitioning the data into training and validation subsets multiple times
  - It helps ensure that the model's performance generalizes well to unseen data
  - Why Use Cross-Validation?
    - Prevents Overfitting
      - By evaluating the model on multiple subsets, cross-validation provides a more robust measure of its performance
    - Reliable Performance Estimate
      - Reduces the variance of performance metrics compared to a single train-test split
    - Optimizes Model Selection
      - Helps in comparing and selecting the best model or hyperparameter configuration.

Types of Cross-Validation

- K-Fold Cross-Validation
  - Splits the dataset into K equal-sized folds
  - Trains the model on K – 1 folds and validates on the remaining fold
  - Repeats the process K times, ensuring each fold is used as a validation set once
  - **Best For:** General-purpose datasets.
- Stratified K-Fold Cross-Validation
  - Ensures that each fold maintains the same class distribution as the original dataset
  - Particularly useful for imbalanced datasets
  - **Best For:** Classification tasks with imbalanced data
- Leave-One-Out Cross-Validation (LOOCV)
  - Uses a single data point as the validation set and the rest as the training set
  - Repeats the process for each data point
  - **Pros:** Maximizes training data for each fold
  - **Cons:** Computationally expensive for large datasets
  - **Best For:** Small datasets where maximizing training data is critical

Practical Guidance on Cross-Validation

- Choose K Based on Dataset Size
  - K = 5 or K = 10 are commonly used for large datasets
  - Use LOOCV for small datasets
- Stratification for Imbalanced Data
  - Always prefer Stratified K-Fold for imbalanced classification tasks to ensure fair evaluation
- Combine with Hyperparameter Tuning
  - Integrate cross-validation into Grid or Random Search for robust hyperparameter tuning

Hands-On Exercise

- Objective
  - Evaluate a classification model using K-Fold and Stratified K-Fold Cross-Validation
  - Compare the results to demonstrate the importance of stratification for imbalanced datasets

Day

06

Automated Hyperparameter Tuning with GridSearchCV and RandomizedSearchCV

Using GridSearchCV and RandomizedSearchCV in scikit-learn

- What is GridSearchCV?
  - Exhaustive search over a specified parameter grid
  - Trains and evaluates a model for every combination of hyperparameters in the grid using cross-validation
- What is RandomizedSearchCV?
  - Selects a fixed number of random combinations from a parameter distribution
  - Faster than GridSearchCV for large hyperparameter spaces while still providing good results

Using GridSearchCV and RandomizedSearchCV in scikit-learn

- Key Features
  - Automates Hyperparameter Tuning
    - Combines model training, evaluation, and hyperparameter search into a single step
  - Cross-Validation Integration
    - Ensures robust performance metrics by using cross-validation
  - Result Interpretation
    - Provides the best hyperparameter combination and associated metrics

Integrating Cross-Validation with Hyperparameter Tuning

- Cross-Validation
  - Ensures that the hyperparameters selected generalize well to unseen data
- Benefits
  - Reduces overfitting to the training dataset
  - Provides robust estimates of model performance

Interpreting Results and Selecting the Best Model

- Best Parameters
  - Access the optimal hyperparameter combination using .best_params_
- Best Estimator
  - Retrieve the model trained with the best hyperparameters using .best_estimator_
- Performance Metrics
  - Use .best_score_ to evaluate the performance of the best hyperparameters.

Hands-On Exercise

- Objective
  - Use GridSearchCV and RandomizedSearchCV to tune hyperparameters of Gradient Boosting and Support Vector Machine models, and compare results

Day

07

Optimization Project – Building and Tuning a Final Model

Applying All Learned Tuning and Optimization Techniques

- Comprehensive Model Optimization
  - Data Preprocessing:
    - Ensure data is clean, scaled, and encoded appropriately
  - Feature Engineering
    - Derive new features and select the most important ones
  - Regularization
    - Avoid overfitting by penalizing complex models
  - Cross-Validation
    - Use techniques like K-Fold or Stratified K-Fold for robust performance metrics
  - Hyperparameter Tuning
    - Use methods like GridSearchCV, RandomizedSearchCV, or Bayesian Optimization

Evaluating and Interpreting Model Performance

- Performance Metrics
  - Classification
    - Accuracy, Precision, Recall, F1-Score, ROC-AUC
  - Regression
    - Mean Squared Error (MSE), Mean Absolute Error (MAE), R^2
  - Importance of Interpretability
    - Use feature importance and coefficient analysis for transparency

Hands-On Project

- Objective
  - Build, tune, and optimize a machine learning model using a structured process and evaluate its performance comprehensively

WEEK 9: Neural Networks and Deep Learning Fundamentals

Day 1 Introduction to Deep Learning and Neural Networks

Day 2 Forward Propagation and Activation Functions

Day 3 Loss Functions and Backpropagation

Day 4 Gradient Descent and Optimization Techniques

Day 5 Building Neural Networks with TensorFlow and Keras

Day 6 Building Neural Networks with PyTorch

Day 7 Neural Network Project – Image Classification on CIFAR-10

Day

01

Introduction to Deep Learning and Neural Networks

What is Deep Learning?

- What is Deep Learning?
  - Subset of Machine Learning that uses artificial neural networks (ANNs) with multiple layers (deep architectures) to model and learn complex patterns in data
  - Key Feature
    - Automatically extracts relevant features from raw data, eliminating the need for manual feature engineering
- Machine Learning vs. Deep Learning

Feature	Machine Learning	Deep Learning
Feature Engineering	Manual feature selection	Automatic feature extraction
Algorithms	Linear regression, SVM, etc.	Neural networks
Data Dependency	Works well with small datasets	Requires large datasets
Computation	Computationally less intensive	Requires GPUs/TPUs for training

Overview of Artificial Neural Networks (ANNs)

- Structure of a Neural Network
  - Input Layer
    - Accepts input data features
  - Hidden Layers
    - Perform computations to extract patterns
  - Output Layer
    - Produces predictions or classifications
- Key Components
  - Neurons
    - Basic units of computation that take inputs, apply weights and biases, and produce outputs using an activation function
  - Weights and Biases
    - Weights determine the importance of each input
    - Bias shifts the output of the activation function
  - Activation Functions
    - Add non-linearity to the model (e.g., ReLU, Sigmoid, Tanh).

Overview of Artificial Neural Networks (ANNs)

- How Neural Networks Work
  - Forward Propagation
    - Data flows through the network to generate predictions
  - Loss Calculation
    - Compares predictions with actual labels to compute the error
  - Backpropagation
    - Adjusts weights and biases using gradient descent to minimize the loss

Applications of Deep Learning

- Domains and Use Cases
  - Computer Vision
    - Image classification (e.g., recognizing objects in images)
    - Object detection (e.g., autonomous vehicles)
  - Natural Language Processing (NLP)
    - Text generation (e.g., GPT models)
    - Sentiment analysis
  - Healthcare
    - Disease diagnosis (e.g., identifying tumors in X-rays)
    - Drug discovery
  - Speech Processing
    - Speech-to-text systems (e.g., virtual assistants)
    - Voice recognition

Hands-On Exercise

- Objective
  - Familiarize yourself with common datasets in deep learning and set up an environment to work with TensorFlow or PyTorch
  - Step 1: Explore Common Datasets
    - MNIST: Handwritten digit dataset (28x28 grayscale images, 10 classes)
    - CIFAR-10: 60,000 32x32 color images across 10 classes
    - ImageNet: Large dataset for image classification with millions of labeled images

Day

02

Forward Propagation and Activation Functions

Understanding Forward Propagation

- What is Forward Propagation?
  - Process by which input data flows through the layers of a neural network to produce an output
  - Input Layer
    - Accepts input features and passes them to the next layer
  - Hidden Layers
    - Compute weighted sums of inputs, apply biases, and pass the result through activation functions
  - Output Layer
    - Produces predictions, typically using an activation function suitable for the task.

Understanding Forward Propagation

- Steps in Forward Propagation
  - Compute Weighted Sum
    - $z = W \cdot X + b$

      W: Weights | X: Inputs | b: Bias
  - Apply Activation Function
    - $a = \sigma(z)$

      σ: Activation function
  - Repeat for Each Layer
    - Outputs of one layer become inputs to the next.

Common Activation Functions

- Sigmoid
  - Use Case: Binary classification in the output layer
  - Limitation: Can suffer from vanishing gradients for large positive/negative z
- Tanh (Hyperbolic Tangent)
  - Use Case: Hidden layers where zero-centered outputs are preferred
  - Limitation: Also prone to vanishing gradients
- ReLU (Rectified Linear Unit)
  - Use Case: Most commonly used in hidden layers due to simplicity and efficiency
  - Limitation: Can suffer from the "dying ReLU" problem (neurons stuck at zero)
- Softmax
  - Use Case: Multi-class classification in the output layer

Choosing Activation Functions

Layer Type	Recommended Activation Function
Hidden Layers	ReLU or Tanh
Output (Binary)	Sigmoid
Output (Multi-Class)	Softmax
Output (Regression)	None or Linear

Hands-On Exercise

- Objective
  - Implement forward propagation for a simple neural network in Python and experiment with different activation functions.

Day

03

Loss Functions and Backpropagation

Understanding Loss Functions

- What Are Loss Functions?
  - Quantify the difference between the predicted output of a model and the actual target value
  - Guide the training process by providing a metric to minimize during optimization
  - Role in Neural Networks
    - Error Measurement
      - Evaluate the accuracy of predictions
    - Feedback for Optimization
      - Provide gradients for weight updates via backpropagation

Understanding Loss Functions

- Common Types of Loss Functions
  - Mean Squared Error (MSE)
    - Commonly used for regression tasks
    - Penalizes larger errors more heavily than smaller ones
  - Cross-Entropy Loss
    - Used for classification tasks
    - Measures the difference between true labels and predicted probabilities

Introduction to Backpropagation

- What Is Backpropagation?
  - Process of computing gradients for each weight and bias in a neural network, enabling optimization algorithms (like gradient descent) to minimize the loss function
  - Steps in Backpropagation
    - Forward Pass: Compute the output and loss for the current weights
    - Backward Pass: Calculate the gradient of the loss with respect to each parameter
    - Weight Update: Use the gradients to update parameters
  - Key Concepts
    - Gradient: The rate of change of the loss with respect to a parameter
    - Gradient Descent: An optimization algorithm that minimizes the loss by updating parameters in the direction of the negative gradient

Hands-On Exercise

- Objective
  - Implement basic loss functions, calculate gradients manually, and visualize the effects of different loss functions

Day

04

Gradient Descent and Optimization Techniques

Gradient Descent and Its Variants

- What is Gradient Descent?
  - Optimization algorithm used to minimize the loss function by iteratively adjusting the model's parameters in the direction of the negative gradient
  - Variants of Gradient Descent
    - Batch Gradient Descent
      - Uses the entire dataset to compute gradients at each step
      - Pros: Accurate gradients
      - Cons: Computationally expensive for large datasets.
    - Stochastic Gradient Descent (SGD)
      - Updates parameters using one data point at a time
      - Pros: Faster updates
      - Cons: High variance in updates; can lead to oscillations.
    - Mini-batch Gradient Descent
      - Updates parameters using a small subset (batch) of the dataset
      - Pros: Combines the efficiency of SGD with the stability of Batch Gradient Descent

Advanced Optimization Techniques

- Adagrad
  - Adapts learning rates for each parameter by scaling inversely with the sum of gradients squared
  - Pros: Suitable for sparse data
  - Cons: Learning rate decreases too aggressively over time
- RMSprop
  - Modifies Adagrad by using an exponentially weighted moving average of squared gradients
  - Pros: Addresses Adagrad's aggressive learning rate decay; works well for non-convex problems
- Adam (Adaptive Moment Estimation)
  - Combines momentum and RMSprop to adapt learning rates for each parameter
  - Pros: Works well in practice for most problems; computationally efficient

Importance of Learning Rate and Choosing the Right Optimizer

- Learning Rate
  - Determines the step size for parameter updates
  - Too High: May overshoot the minimum or cause divergence
  - Too Low: Leads to slow convergence
- Choosing the Right Optimizer
  - SGD: Works well for simple, convex problems
  - Adam: Generally performs well across tasks
  - RMSprop: Often preferred for RNNs and sequence-based tasks

Hands-On Exercise

- Objective
  - Implement gradient descent to update model weights and experiment with different optimizers using TensorFlow and/or PyTorch

Day

05

Building Neural Networks with TensorFlow and Keras

Introduction to TensorFlow and Keras

- What is TensorFlow?
  - Open-source library for numerical computation and machine learning
  - Provides tools for building and training deep learning models
- What is Keras?
  - High-level API integrated with TensorFlow that simplifies the process of creating and training neural networks
  - Key Features of Keras
    - User-Friendly: Intuitive syntax for rapid prototyping
    - Modular: Building blocks for defining layers, optimizers, and loss functions
    - Integration: Compatible with TensorFlow for scalable deep learning tasks

Defining Layers, Models, and Compiling Networks in Keras

- Defining Layers
  - Layers are the building blocks of neural networks. Common types include:
    - Dense (Fully Connected) Layers
      - Each neuron is connected to every neuron in the previous layer
    - Dropout Layers
      - Randomly drops connections to prevent overfitting
    - Activation Layers
      - Apply activation functions to introduce non-linearity

Defining Layers, Models, and Compiling Networks in Keras

- Building a Model
  - Keras supports two primary ways to define models
    - Sequential API: A linear stack of layers
    - Functional API: More flexible, allows for complex architectures
- Compiling a Model
  - Specifies
    - Optimizer: Algorithm to update weights
    - Loss Function: Metric to minimize during training
    - Metrics: Additional performance measures

Training, Evaluating, and Saving a Model

- Training
  - Fit the model to data using model.fit()
- Evaluation
  - Test the model on unseen data using model.evaluate()
- Saving and Loading
  - Save a trained model using model.save() and reload it with keras.models.load_model()

Hands-On Exercise

- Objective
  - Build, train, evaluate, and save a simple neural network to classify digits from the MNIST dataset

Day

06

Building Neural Networks with PyTorch

Introduction to PyTorch and Its Core Components

- What is PyTorch?
  - Open-source deep learning framework that provides flexibility and dynamic computation for building and training machine learning models
  - Core Components of PyTorch
    - Tensors: Multi-dimensional arrays similar to NumPy arrays but with GPU support for accelerated computation
    - Autograd: Automatic differentiation engine that computes gradients for optimization
    - ***torch.nn*** Module: Provides tools to define and train neural networks with layers, activation functions, and loss functions.

Building a Neural Network in PyTorch

- Steps
  - Define the Model
    - Use torch.nn.Module to create a neural network with layers and forward propagation
  - Define the Loss Function
    - Use built-in loss functions like Cross-Entropy Loss
  - Define the Optimizer
    - Use optimizers like SGD or Adam for weight updates

Training, Evaluating, and Saving a Model in PyTorch

- Training
  - Forward pass to compute predictions
  - Compute loss and gradients using backpropagation
  - Update weights using an optimizer
- Evaluation
  - Test the model on unseen data and calculate metrics like accuracy
- Saving and Loading
  - Save the model's parameters using torch.save() and load them using torch.load()

Hands-On Exercise

- Objective
  - Build, train, evaluate, and save a neural network for MNIST digit classification using PyTorch.

Day

07

Neural Network Project – Image Classification on CIFAR-10

Applying Learned Concepts to a More Complex Dataset

- Why CIFAR-10?
  - The CIFAR-10 dataset is a challenging benchmark dataset for image classification. It contains 60,000 32x32 color images across 10 classes (e.g., airplane, car, bird, dog)
  - Unlike MNIST, CIFAR-10 involves more complex patterns and requires robust neural network architectures

Building and Optimizing a Neural Network for Image Classification

- Key Steps:
  - Preprocess the dataset for training (e.g., normalization, one-hot encoding)
  - Define a neural network with convolutional layers for feature extraction
  - Optimize the network using techniques like learning rate adjustment and dropout.

Analyzing Model Performance and Experimenting with Hyperparameters

- Performance Analysis
  - Evaluate accuracy and loss curves during training
  - Use test set metrics to measure generalization
- Experimentation
  - Try different activation functions (e.g., ReLU, Tanh)
  - Test optimizers like SGD, Adam, and RMSprop
  - Adjust the learning rate and regularization techniques (e.g., dropout, weight decay).

Hands-On Project

- Objective
  - Build, train, and optimize a neural network for CIFAR-10 image classification, experimenting with hyperparameters to improve performance

WEEK 10: Convolutional Neural Networks (CNNs)

Day 1 Introduction to Convolutional Neural Networks

Day 2 Convolutional Layers and Filters

Day 3 Pooling Layers and Dimensionality Reduction

Day 4 Building CNN Architectures with Keras and TensorFlow

Day 5 Building CNN Architectures with PyTorch

Day 6 Regularization and Data Augmentation for CNNs

Day 7 CNN Project – Image Classification on Fashion MNIST or CIFAR-10

Day

01

Introduction to Convolutional Neural Networks

Overview of CNNs and Their Role in Image Processing

- What are Convolutional Neural Networks (CNNs)?
  - Specialized type of neural network designed for processing structured grid data, such as images
  - Particularly effective for image-related tasks like classification, object detection, and segmentation
  - Why CNNs for Image Processing?
    - Spatial Hierarchies
      - CNNs capture spatial and hierarchical patterns in images
      - Convolutional layers extract features like edges, textures, and complex structures
    - Parameter Efficiency
      - Unlike fully connected networks, CNNs use fewer parameters due to shared weights, reducing computation and memory requirements

CNN Architecture

- Key Components of a CNN
  - Convolutional Layers
    - Perform convolution operations to extract features
    - Kernel/Filter
      - A small matrix (e.g., 3x3) that slides over the input image to detect patterns
    - Output
      - Feature maps highlighting specific patterns in the input
  - Pooling Layers
    - Downsample feature maps to reduce dimensions and computation
    - Types
      - Max Pooling: Takes the maximum value in a region
      - Average Pooling: Takes the average value in a region
  - Fully Connected Layers
    - Combine extracted features for final predictions
    - Act as a "classifier" in the network
  - Basic CNN Workflow: Input Image → Convolution → Activation → Pooling → Fully Connected Layer → Output

Key Advantages of CNNs Over Fully Connected Networks for Images

- Translation Invariance
  - CNNs can detect patterns irrespective of their position in the image
- Reduced Parameters
  - Shared weights and local connectivity make CNNs computationally efficient
- Automatic Feature Extraction
  - CNNs learn to identify meaningful patterns like edges, shapes, and textures directly from data

Hands-On Exercise

- Objective
  - Visualize images in a dataset, explore their pixel data, and set up an environment for building CNNs using TensorFlow or PyTorch

Day

02

Convolutional Layers and Filters

Convolution Operations, Filters, and Feature Maps

- What is a Convolution Operation?
  - Mathematical operation where a small matrix (kernel or filter) slides over the input image to extract features like edges, textures, or patterns
  - Key Concepts
    - Kernel (Filter)
      - A small matrix (e.g., 3×3 3×3) used to extract features
      - Each element of the kernel is a weight learned during training
    - Feature Map
      - The output of a convolution operation
      - Highlights specific patterns detected by the filter
    - Channels
      - For RGB images, convolution processes each color channel separately and combines results

Concepts of Kernel Size, Stride, and Padding

- Kernel Size
  - The dimensions of the filter (e.g., 3×3 3×3, 5×5 5×5)
  - Smaller Kernels: Capture fine details
  - Larger Kernels: Detect broader features
- Stride
  - Defines the step size of the filter as it slides across the input
  - Larger Strides: Reduce the feature map size, improving computation efficiency
  - Smaller Strides: Retain more detail but increase computation
- Padding
  - Adds extra pixels around the input to control the size of the output
  - Valid Padding: No padding; the feature map shrinks
  - Same Padding: Adds enough padding to keep the output size equal to the input size

Visualizing How Convolution Extracts Features

- Edge Detection
  - Kernels like Sobel or Prewitt highlight edges in images
- Feature Extraction
  - Initial layers focus on edges; deeper layers capture abstract patterns.

Hands-On Exercise

- Objective
  - Understand convolution operations by implementing and visualizing their effects using TensorFlow and PyTorch

Day

03

Pooling Layers and Dimensionality Reduction

Introduction to Pooling Layers

- What Are Pooling Layers?
  - Used to reduce the dimensions of feature maps while retaining the most important information
  - Help make the network computationally efficient and robust to variations in the input
  - Types of Pooling
    - Max Pooling
      - Selects the maximum value from each region of the input feature map
      - Captures the strongest activations (features)
    - Average Pooling
      - Computes the average value for each region of the input feature map
      - Provides a more generalized summary of features.

Role of Pooling in Reducing Dimensionality

- Dimensionality Reduction
  - Pooling reduces the spatial dimensions (height and width) of feature maps, resulting in fewer parameters and faster computations
- Robustness
  - Makes the model invariant to small translations or distortions in the input image

Combining Convolution and Pooling Layers

- Pooling layers typically follow convolutional layers to downsample the feature maps
- This combination helps extract hierarchical features
  - Early layers focus on simple features (e.g., edges)
  - Deeper layers capture complex patterns (e.g., objects)

Hands-On Exercise

- Objective
  - Implement max pooling and average pooling layers on feature maps and observe their effects on size and representation.

Day

04

Building CNN Architectures with Keras and TensorFlow

Building a CNN Architecture in Keras

- Steps to Build a CNN
  - Convolutional Layers: Extract features from the input images
  - Pooling Layers: Downsample feature maps to reduce dimensions and retain key features
  - Dense (Fully Connected) Layers: Combine features for final predictions
- Basic CNN Architecture
  - Input Layer → Convolutional Layer → Activation → Pooling → Fully Connected Layer → Output Layer
  - Repeat convolution and pooling layers for deeper networks

Compiling, Training, and Evaluating a CNN

- Steps
  - Compile the Model
    - Define loss, optimizer, and metrics
    - Example loss functions
      - Categorical Cross-Entropy: Multi-class classification
    - Example optimizers
      - Adam: Efficient optimization for large networks
    - Example metrics: Accuracy
  - Train the Model
    - Use model.fit() with training data, validation data, epochs, and batch size
  - Evaluate the Model
    - Use model.evaluate() with test data to calculate metrics

Introduction to Popular CNN Architectures

- LeNet
  - One of the earliest CNNs for handwritten digit classification (e.g., MNIST)
- AlexNet
  - Revolutionized deep learning for image classification in 2012
  - Introduced ReLU activation and dropout for regularization
- VGG
  - Uses deep networks with small filters (e.g., 3×3 3×3)
  - Known for its simplicity and effectiveness

Hands-On Exercise

- Objective
  - Build, train, and evaluate a CNN for image classification on the MNIST or CIFAR-10 dataset using Keras and TensorFlow

Day

05

Building CNN Architectures with PyTorch

Building CNN Architectures in PyTorch Using the nn Module

- Key Steps
  - Define a Model
    - Use **torch.nn.Module** to build CNN layers like convolutional, pooling, and fully connected layers
  - Forward Pass
    - Define how input flows through the layers to produce output
  - Model Summary
    - Inspect the structure and learnable parameters

Training and Evaluating CNNs in PyTorch

- Training
  - Perform forward and backward passes, calculate loss, and update weights using an optimizer
- Evaluation
  - Test the model on unseen data and compute metrics like accuracy and loss.

Experimenting with CNN Model Design and Tuning Hyperparameters

- Experimentation Areas
  - Layer Depth
    - Add or remove convolutional and pooling layers to observe the impact
  - Filter Size
    - Experiment with kernel sizes (e.g., 3×3, 5×5)
  - Learning Rate
    - Adjust the learning rate to improve convergence speed and accuracy

Hands-On Exercise

- Objective
  - Build, train, evaluate, and experiment with CNNs for CIFAR-10 classification using PyTorch

Day

06

Regularization and Data Augmentation for CNNs

Overfitting in CNNs and Methods to Prevent It

- What Is Overfitting?
  - Occurs when a model performs well on the training data but fails to generalize to unseen data
  - In CNNs, overfitting is common due to the large number of parameters in deep networks
  - Methods to Prevent Overfitting
    - Dropout
      - Randomly sets a fraction of neurons to zero during training
      - Prevents co-adaptation of neurons
      - Controlled by a dropout rate (e.g., 0.5)
    - Batch Normalization
      - Normalizes the input of each layer to stabilize training
      - Reduces internal covariate shift and allows higher learning rates
    - Data Augmentation
      - Increases dataset size artificially by applying transformations to images
      - Examples: rotation, flipping, scaling, cropping, brightness adjustment.

Introduction to Data Augmentation Techniques

- Common Techniques
  - Rotation
    - Rotates the image by a specified angle range (e.g., -30° to 30°)
  - Flipping
    - Horizontally or vertically flips the image
  - Scaling
    - Resizes the image by zooming in or out
  - Cropping
    - Extracts random portions of the image

Implementing Regularization and Data Augmentation in CNN Training

- Why Use Both?
  - Regularization reduces the complexity of the model
  - Data augmentation increases the diversity of the training data, improving generalization

Hands-On Exercise

- Objective
  - Apply dropout, batch normalization, and data augmentation to improve CNN performance

Day

07

CNN Project
Image Classification on Fashion MNIST or CIFAR-10

Applying CNN Architecture to a Larger Dataset

- Why Larger Datasets?
  - Larger datasets like CIFAR-10 or Fashion MNIST represent more realistic and diverse challenges compared to toy datasets like MNIST
  - They require deeper architectures, careful regularization, and augmentation for optimal performance

Experimenting with Architecture Design, Regularization, and Augmentation

- Key Techniques to Improve Performance
  - Architectural Modifications
    - Add more convolutional layers or change kernel sizes
    - Use more filters in deeper layers to capture complex features
  - Regularization
    - Apply dropout in dense layers and batch normalization in convolutional layers
    - Prevent overfitting in deeper models
  - Data Augmentation
    - Use techniques like random flipping, cropping, and rotation to improve generalization

Analyzing Model Performance and Tuning

- Evaluation Metrics
  - Accuracy: Overall classification correctness
  - Loss: Measures the difference between predictions and ground truth
  - Confusion Matrix: Highlights misclassified classes for deeper insights

Hands-On Exercise

- Objective
  - Build, train, and optimize a CNN for Fashion MNIST or CIFAR-10 image classification, experimenting with regularization and data augmentation to achieve the best performance

WEEK 11: Recurrent Neural Networks (RNNs) and Sequence Modeling

Day 1 Introduction to Sequence Modeling and RNNs

Day 2 Understanding RNN Architecture and Backpropagation Through Time (BPTT)

Day 3 Long Short-Term Memory (LSTM) Networks

Day 4 Gated Recurrent Units (GRUs)

Day 5 Text Preprocessing and Word Embeddings for RNNs

Day 6 Sequence-to-Sequence Models and Applications

Day 7 RNN Project – Text Generation or Sentiment Analysis

Day

01

Introduction to Sequence Modeling and RNNs

Overview of Sequence Modeling

- What is Sequence Modeling?
  - Involves predicting or generating outputs based on sequential data
  - Capture temporal or contextual dependencies
  - Why Is Sequence Modeling Important?
    - Natural Language Processing (NLP)
      - Tasks like language modeling, machine translation, and sentiment analysis depend on understanding sequential relationships in text
    - Time-Series Analysis
      - Sequence models are essential for tasks like stock price prediction, weather forecasting, and sensor data analysis

Introduction to Recurrent Neural Networks (RNNs)

- What Are RNNs?
  - Specialized neural networks for sequence modeling
  - They include recurrent connections that allow them to maintain a "memory" of previous inputs
  - Structure of an RNN
    - Input Layer
      - Processes sequential data one step at a time
    - Hidden State
      - Maintains information about past inputs
    - Recurrent Connections
      - Pass hidden states from one time step to the next

Introduction to Recurrent Neural Networks (RNNs)

- Key Concepts in RNNs:
  - Hidden States
    - Represent memory at each time step
  - Recurrent Connections
    - Enable information sharing across time steps
  - Vanishing Gradient Problem
    - Challenges in training long sequences due to diminishing gradients

Key Applications of RNNs in Machine Learning

- Text Generation
  - Generate new text based on learned patterns
- Language Translation
  - Convert text from one language to another
- Stock Price Prediction
  - Predict future stock prices using historical data
- Speech Recognition
  - Understand spoken words and phrases
- Video Frame Prediction
  - Anticipate future frames in video sequences

Hands-On Exercise

- Objective
  - Preprocess a text dataset for use in RNNs and set up an environment in TensorFlow or PyTorch for building RNNs

Day

02

Understanding RNN Architecture and Backpropagation Through Time (BPTT)

Detailed Architecture of RNNs

- Components of an RNN
  - Input Layer
    - Takes sequential data as input at each time step
  - Hidden Layer
    - Maintains a "memory" of past inputs through recurrent connections. The hidden state at time t(h_t) is calculated as
      - $h_t = f(W_h \cdot h_{t-1} + W_x \cdot x_t + b_h)$
        - W_h : Weight matrix for recurrent connections
        - W_x : Weight matrix for input connections
        - b_h : Bias term
        - f : Non-linear activation function (e.g., tanh, ReLU)
  - Output Layer
    - Produces output y_t based on the hidden state h_t
      - $y_t = g(W_y \cdot h_t + b_y)$
        - g: Activation function (e.g., softmax for classification)

Backpropagation Through Time (BPTT)

- **What is BPTT?**
  - Extension of standard backpropagation to handle sequential data in RNNs
  - It calculates gradients for each time step and propagates them backward through the sequence
  - **Steps of BPTT**
    - Unroll the RNN across the sequence for a fixed number of time steps
    - Compute the loss for each time step
    - Backpropagate the errors across all time steps to update weights
  - **Challenges in BPTT**
    - Vanishing Gradient Problem
      - Gradients diminish exponentially as they are propagated back through time
      - Leads to difficulty in learning long-term dependencies
    - Exploding Gradient Problem
      - Gradients grow exponentially, causing numerical instability during training
  - **Solutions**
    - Use gradient clipping to handle exploding gradients
    - Use architectures like Long Short-Term Memory (LSTM) or Gated Recurrent Units (GRU) to mitigate the vanishing gradient problem

Limitations of Vanilla RNNs

- Short-Term Memory
  - Struggle to learn dependencies in long sequences due to vanishing gradients
- Sequential Computation
  - Cannot parallelize training across time steps, making them computationally expensive
- Sensitive Initialization
  - Performance depends heavily on proper weight initialization and learning rates

Hands-On Exercise

- Objective
  - Build a simple RNN model for text classification using TensorFlow or PyTorch
  - Train the RNN and observe how it captures sequence patterns

Day

03

Long Short-Term Memory (LSTM) Networks

Introduction to LSTMs and How They Address RNN Limitations

- What Are LSTMs?
  - Type of Recurrent Neural Network (RNN) specifically designed to handle long-term dependencies
  - LSTMs mitigate the vanishing gradient problem by using specialized gates to manage the flow of information
  - Key Features of LSTMs
    - Memory Cells
      - Maintain a long-term memory state across sequences
    - Gated Mechanism
      - Regulates how much information to keep, update, or forget at each time step
    - Effective for Long Sequences
      - Handles sequential data with dependencies across many time steps.

Introduction to LSTMs and How They Address RNN Limitations

- Advantages Over Vanilla RNNs
  - Retains long-term dependencies
  - Prevents gradient-related issues during training
  - Outperforms RNNs on tasks like language modeling, speech recognition, and time-series forecasting

LSTM Cell Structure: Input, Forget, and Output Gates

- Forget Gate
  - Decides what information to discard from the cell state
    - $f_t = \sigma\ (W_f\ .\text{dot}\ [h_{t-1}, x_t] + b_f)\ ]$
      - W_f: Weight matrix for the forget gate
      - f_t: Forget gate output
      - x_t: Input
      - h_{t-1} : Previous hidden state
- Input Gate
  - Decides what new information to add to the cell state
    - $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
      - σ: Sigmoid activation function
      - W_i : Weight matrix for the input gate
      - h_{t-1} : Hidden state from the previous time step
      - x_t : Current input
      - b_i: Bias for the input gate

LSTM Cell Structure: Input, Forget, and Output Gates

- Cell State Update
  - Combines the forget gate and input gate results to update the cell state
    - $C_t = f_t$. dot $C_{t-1} + i_t$. dot $C_{t'}$
- Output Gate
  - Decides what information to output at each time step
    - $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
      - W_o: Weight matrix for the output gate
      - b_o : Bias for the output gate

Applications of LSTMs

- Natural Language Processing (NLP)
  - Sentiment analysis, machine translation, text generation
- Time-Series Forecasting
  - Predicting stock prices, weather patterns, or sales trends
- Speech Recognition
  - Converting spoken words into text
- Anomaly Detection
  - Identifying unusual patterns in sequential data

Hands-On Exercise

- Objective
  - Build an LSTM model for sentiment analysis on the IMDB Movie Reviews Dataset and compare its performance with a basic RNN model

Day

04

Gated Recurrent Units (GRUs)

Introduction to Gated Recurrent Units (GRUs)

- What Are GRUs?
  - Simplified variant of Long Short-Term Memory (LSTM) networks
  - Designed to retain long-term dependencies while reducing computational complexity by having fewer parameters
  - Key Features of GRUs
    - Simpler Architecture
      - GRUs have two gates (update and reset) compared to LSTMs' three gates (input, forget, output)
    - Efficiency
      - Fewer parameters make GRUs computationally faster and less prone to overfitting on smaller datasets
    - Retains Performance
      - Comparable to LSTMs in terms of capturing sequential dependencies.

GRU Cell Structure: Update and Reset Gates

- Update Gate
  - Determines how much of the previous hidden state to retain and how much to update with new information
- Reset Gate
  - Controls how much of the past information to forget when combining with new input.
- Hidden State Update
  - Combines the new information and the past hidden state based on the reset and update gates

When to Use GRUs vs. LSTMs

Feature	LSTM	GRU
Gates	Input, Forget, Output	Update, Reset
Parameters	More	Fewer
Performance	Better for complex, longer sequences	Comparable for shorter sequences
Training Speed	Slower due to complexity	Faster due to simpler structure
Use Cases	NLP, speech recognition	Time-series data, small datasets

Hands-On Exercise

- Objective
  - Build a GRU-based model for the IMDB Movie Reviews Dataset and compare its performance with the LSTM model

Day

05

Text Preprocessing and Word Embeddings for RNNs

Importance of Text Preprocessing

- What is Text Preprocessing?
  - Involves cleaning and preparing raw text data to make it suitable for machine learning models
  - Critical step for achieving high performance in Natural Language Processing (NLP) tasks
  - Key Steps
    - Tokenization: Splits text into individual units (e.g., words, sentences)
      - Example: "I love NLP" → ["I", "love", "NLP"]
    - Stemming: Reduces words to their root form by removing suffixes
      - Example: "running", "runner" → "run"
    - Lemmatization: Converts words to their base form using a vocabulary
      - Example: "better" → "good"
  - Why is Preprocessing Important?
    - Reduces noise in the data
    - Standardizes input for models
    - Improves feature extraction and model accuracy

Introduction to Word Embeddings

- What Are Word Embeddings?
  - Dense vector representations of words that capture semantic meaning
  - Represent words in a continuous vector space
  - Popular Word Embedding Models
    - Word2Vec:
      - Models: Continuous Bag of Words (CBOW) and Skip-gram
      - Captures word relationships based on context
    - GloVe (Global Vectors for Word Representation)
      - Uses word co-occurrence statistics to generate embeddings
      - Represents global semantic relationships
    - Pre-trained Embeddings in Frameworks
      - Frameworks like TensorFlow and PyTorch offer pre-trained embeddings for quick integration
  - Benefits of Word Embeddings
    - Reduce dimensionality | Capture semantic similarity | Improve model generalization

Using Pre-trained Embeddings for NLP Tasks

- Why Use Pre-trained Embeddings?
  - Saves computational resources
  - Leverages large, well-trained models
  - Boosts performance for downstream tasks
- Popular Pre-trained Embeddings
  - GloVe: Pre-trained on datasets like Wikipedia and Common Crawl
  - FastText: Handles out-of-vocabulary (OOV) words through subword embeddings
  - Embedding Layers in deep learning frameworks like TensorFlow and PyTorch

Hands-On Exercise

- Objective
  - Preprocess a text dataset and integrate word embeddings (e.g., GloVe) into an LSTM model for sentiment analysis

Day

06

Sequence-to-Sequence Models and Applications

Sequence-to-Sequence (Seq2Seq) Models and Their Architecture

- What Are Seq2Seq Models?
  - Map an input sequence to an output sequence of different lengths
  - Widely used for tasks like language translation, text summarization, speech-to-text, and chatbots
  - Architecture
    - Encoder
      - Processes the input sequence and encodes it into a fixed-length vector (context vector)
    - Decoder
      - Takes the context vector as input and generates the output sequence, step by step

Encoder-Decoder Framework for Seq2Seq Tasks

- How It Works
  - Encoder
    - Sequentially processes the input sequence using RNN, LSTM, or GRU
    - Produces a context vector representing the entire input sequence
  - Decoder
    - Initializes its hidden state with the encoder's context vector
    - Generates the output sequence one token at a time
    - Predicts the next token using the previously generated tokens

Attention Mechanism Overview

- Why Attention?
  - Standard Seq2Seq models compress the entire input sequence into a fixed-length vector, which can lead to information loss for long sequences
  - Attention Mechanism dynamically focuses on different parts of the input sequence when generating each output token
- How Attention Works
  - Calculates a weight (or score) for each input token based on its relevance to the current decoder state
  - Outputs a weighted sum of the encoder outputs, creating a context vector for each decoder step

Hands-On Exercise

- Objective
  - Build a basic Seq2Seq model using LSTMs for translation and experiment with hyperparameters

Day

07

RNN Project
Sentiment Analysis

Applying RNN, LSTM, and GRU Models to a Complete Task

- Project Overview
  - Choose a task: Sentiment Analysis
    - Sentiment Analysis: Classify text into categories like positive or negative sentiment
  - Use RNN, LSTM, or GRU models to solve the chosen task
  - Focus on preprocessing, embedding integration, model architecture, and hyperparameter tuning

Analyzing Model Performance

- Key Metrics
  - For Sentiment Analysis
    - Accuracy, Precision, Recall, F1 Score
- Tuning Hyperparameters
  - Embedding size
  - Number of hidden units
  - Learning rate and optimizer choice
  - Sequence length and batch size

Experimenting with Architectures and Techniques

- Architectural Variations
  - Single-Layer vs. Multi-Layer RNNs
  - Bidirectional RNNs for improved context capture
- Preprocessing Techniques
  - Tokenization, stemming, lemmatization
  - Padding for sequence uniformity

Hands-On Project

- Objective
  - Build, train, and optimize RNN, LSTM, or GRU models for Sentiment Analysis

WEEK 12: Transformers and Attention Mechanisms

Day 1 Introduction to Attention Mechanisms

Day 2 Introduction to Transformers Architecture

Day 3 Self-Attention and Multi-Head Attention in Transformers

Day 4 Positional Encoding and Feed-Forward Networks

Day 5 Hands-On with Pre-Trained Transformers – BERT and GPT

Day 6 Advanced Transformers – BERT Variants and GPT-3

Day 7 Transformer Project – Text Summarization or Translation

Day

01

Introduction to Attention Mechanisms

Understanding the Limitations of RNNs and the Need for Attention

- Challenges of RNNs
  - Sequential Processing
  - Long-Term Dependency Problems
  - Fixed Context Vector
- The Role of Attention Mechanisms
  - Attention overcomes these limitations by allowing the model to focus on specific parts of the input sequence dynamically during each output generation step
  - Instead of relying on a single context vector, attention provides a weighted combination of all input tokens relevant to the current output token

Basics of the Attention Mechanism

- Core Components
  - Queries (Q)
    - Represents the current focus of the model (e.g., the current decoder state in Seq2Seq tasks)
  - Keys (K)
    - Encoded representations of the input sequence
  - Values (V)
    - Additional information associated with the keys.

Basics of the Attention Mechanism

- Attention Mechanism
  - The attention score is computed using the dot product of the query and keys, followed by a softmax function to normalize into a probability distribution
  - The weighted sum of the values forms the context vector

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Types of Attention

- Self-Attention
  - The query, key, and value all come from the same input sequence
  - Widely used in Transformer models for learning interdependencies within a sequence
- Multi-Head Attention
  - Extends self-attention by applying multiple attention mechanisms in parallel
  - Captures different aspects of relationships in the sequence

$$MultiHead(Q, K, V) = Concat(head_1, head_2, \ldots, head_h)W^O$$

    - where each head computes attention with different learned projections of Q, K, and V

Hands-On Exercise

- Objective
  - Implement a basic Attention Mechanism using NumPy or PyTorch and visualize its impact on a simple sequence task

Day

02

Introduction to Transformers Architecture

Overview of the Transformer Architecture

- What is a Transformer?
  - Neural network architecture introduced in the paper "Attention is All You Need”
  - It relies entirely on the attention mechanism to process sequential data without using recurrence or convolution
  - Transformative for NLP tasks like translation, summarization, and text generation

Overview of the Transformer Architecture

- Components of the Transformer
  - Encoder
    - Processes the input sequence and generates a contextualized representation
    - Consists of multiple identical layers, each with
      - Self-Attention Mechanism: Captures dependencies between all input tokens
      - Feed-Forward Neural Network (FFNN): Processes the attention outputs
  - Decoder
    - Generates the output sequence one token at a time
    - Consists of multiple identical layers, each with
      - Masked Self-Attention Mechanism: Prevents the decoder from attending to future tokens
      - Encoder-Decoder Attention: Attends to encoder outputs
      - Feed-Forward Neural Network
  - Workflow: Input sequence → Encoder → Context vectors → Decoder → Output sequence

Detailed Breakdown of the Transformer Model Layers

- Self-Attention Layer
  - Captures relationships between all tokens in the input sequence
  - Computes the importance of each token to all other tokens
- Positional Encoding
  - Since Transformers lack recurrence, positional encoding injects information about the token order into the model
- Feed-Forward Neural Network
  - Applies a position-wise FFNN to the outputs of the attention layer
  - Non-linear transformation enhances the representation
- Layer Normalization
  - Stabilizes training by normalizing inputs within each layer
- Multi-Head Attention
  - Combines multiple self-attention mechanisms to learn various aspects of relationships within the sequence

Key Differences Between Transformers and RNNs

Aspect	Transformers	RNNs
Parallelization	Fully parallelizable; faster training	Sequential processing limits parallelization
Long-Term Dependencies	Handles long dependencies via attention	Struggles with long dependencies
Position Information	Uses positional encodings	Implicitly encodes position
Architecture	Built on attention mechanisms; no recurrence	Sequential processing through hidden states

Hands-On Exercise

- Objective
  - Visualize the architecture of a Transformer model and set up an environment for working with Transformers using PyTorch and/or TensorFlow

Day

03

Self-Attention and Multi-Head Attention in Transformers

Self-Attention Mechanism

- What is Self-Attention?
  - Allows a model to dynamically focus on different parts of an input sequence when encoding a token
  - It captures dependencies across all tokens in a sequence, enabling context-aware representations
  - Steps in Self-Attention
    - Compute Attention Scores:
      - Calculate dot products between the query (Q) and key (K) vectors for all tokens
      - Scale by the square root of the key dimension (d_k) to stabilize gradients
      - Apply the softmax function to convert scores into probabilities
    - Weight Values
      - Use the attention scores to compute a weighted sum of value (V) vectors

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Multi-Head Attention

- What is Multi-Head Attention?
  - Applies several attention mechanisms in parallel
  - Each attention "head" focuses on different aspects of the sequence
- Steps
  - Linear Projections
    - Project Q, K, and V into multiple subspaces using learned weight matrices
  - Apply Self-Attention
    - Perform self-attention for each head independently
  - Concatenate Outputs
    - Combine outputs from all heads
  - Final Linear Projection
    - Project concatenated outputs back into the original dimension

$$\textbf{MultiHead}(Q, K, V) = \textbf{Concat}(\textbf{head}_1, \textbf{head}_2, \ldots, \textbf{head}_h) W_O$$

Applications of Multi-Head Attention in NLP

- Machine Translation
  - Captures dependencies across languages for better translations
- Text Summarization
  - Identifies key phrases to generate concise summaries
- Named Entity Recognition
  - Focuses on contextual clues to detect entities in text

Hands-On Exercise

- Objective
  - Implement a simplified Self-Attention and Multi-Head Attention mechanism and visualize their effects on text sequences

Day

04

Positional Encoding and Feed-Forward Networks

Understanding the Role of Positional Encoding in Transformers

- Why Positional Encoding?
  - Unlike RNNs, Transformers do not process sequences sequentially
  - They process all tokens in parallel
  - Transformers lack inherent knowledge of token positions, which is crucial for tasks like translation or sequence modeling
- What is Positional Encoding?
  - Positional encoding introduces information about the order of tokens in a sequence
  - It allows the model to differentiate between identical tokens in different positions.

Mathematical Foundation and Implementation of Positional Encoding

- Sinusoidal Positional Encoding
  - Encodes positional information using sine and cosine functions
  - Formula for positional encoding

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

Where: pos: Position of the token in the sequence
i: Index of the embedding dimension
d: Total embedding dimension

- Why Sinusoidal Functions?
  - Provides unique encoding for each position.
  - Allows generalization to longer sequences not seen during training

The Feed-Forward Network

- What is a Feed-Forward Network (FFN)?
  - FFNs are fully connected layers applied to each token independently and identically within a Transformer layer
  - Adds non-linear transformation to the output of the attention mechanism
- Role in Transformers
  - Captures token-specific transformations
  - Enhances representational capacity
- Structure
  - Linear transformation
  - Non-linear activation (e.g., ReLU)
  - Another linear transformation

$$FFN(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

Hands-On Exercise

- Objective
  - Implement positional encoding and integrate it with a basic Transformer model
  - Experiment with different positional encoding methods and observe the effects

Day

05

Hands-On with Pre-Trained Transformers BERT and GPT

Introduction to BERT and GPT

- What is BERT?
  - BERT (Bidirectional Encoder Representations from Transformers)
    - Developed by Google AI
    - Processes input sequences bidirectionally, enabling it to capture context from both directions
    - Pre-trained on tasks like Masked Language Modeling (MLM) and Next Sentence Prediction (NSP)
  - Key Features of BERT
    - Bidirectional: Understands context from both left and right sides of a word
    - Transformer Encoder-Based: Optimized for understanding input text
    - Applications: Sentiment analysis, named entity recognition, question answering

What is GPT?

- GPT (Generative Pretrained Transformer)
  - Developed by OpenAI
  - Processes input sequences unidirectionally (left-to-right), focusing on generative tasks
  - Pre-trained using causal language modeling
- Key Features of GPT
  - Unidirectional: Processes text from left to right, focusing on text generation
  - Transformer Decoder-Based: Optimized for generating coherent text
  - Applications: Text generation, chatbots, summarization

Key Differences Between BERT and GPT

Aspect	BERT	GPT
Architecture	Transformer Encoder	Transformer Decoder
Training Objective	MLM, NSP	Causal Language Modeling
Directionality	Bidirectional	Unidirectional
Use Cases	Understanding tasks (e.g., classification)	Generative tasks (e.g., text generation)

Fine-Tuning Pre-Trained Models for Downstream Tasks

- Why Fine-Tune?
  - Pre-trained models are trained on large generic datasets
  - Fine-tuning adapts them to specific tasks like sentiment analysis or classification
- Steps to Fine-Tune
  - Load a Pre-Trained Model
    - Use libraries like Hugging Face to load a pre-trained BERT or GPT model
  - Prepare Dataset
    - Format the dataset for the specific task (e.g., tokenization for text classification)
  - Train and Evaluate
    - Fine-tune the model using task-specific data

Hands-On Exercise

- Objective
  - Use Hugging Face's Transformers library to fine-tune a pre-trained BERT or GPT model for a text classification task

Day

06

Advanced Transformers BERT Variants and GPT-3

Exploration of BERT Variants

- Why BERT Variants?
  - While BERT is powerful, it has limitations like large computational requirements and inefficiencies in capturing certain nuances
  - BERT variants optimize the model for specific tasks, improve performance, or reduce computational overhead
  - Key BERT Variants
    - RoBERTa (Robustly Optimized BERT)
      - Removes Next Sentence Prediction (NSP) task for better efficiency
      - Trains on more data with larger batch sizes
      - Use Case: Superior performance in tasks requiring deeper context
    - DistilBERT
      - A distilled (smaller) version of BERT that retains 97% of BERT's performance while being 60% faster
      - Use Case: Ideal for real-time applications and resource-constrained environments
    - ALBERT (A Lite BERT)
      - Reduces memory consumption by factorizing embeddings and sharing parameters across layers
      - Use Case: Suitable for large-scale pre-training and downstream tasks with memory limitations
    - BERTweet
      - Fine-tuned on Twitter data
      - Use Case: Social media sentiment analysis, hashtag prediction.

Introduction to GPT-3

- What is GPT-3?
  - GPT-3 (Generative Pretrained Transformer 3)
    - Developed by OpenAI
    - A massive model with 175 billion parameters trained on diverse datasets
    - Excels at generating coherent and contextually relevant text
    - Key Features of GPT-3
      - Zero-shot and Few-shot Learning
        - Can perform tasks with minimal or no fine-tuning
      - Versatility
        - Used for text generation, summarization, question answering, and conversational AI
    - Applications
      - Conversational AI: Chatbots and virtual assistants
      - Content Generation: Articles, scripts, code snippets
      - Creative Writing: Poems, stories, and creative ideas

Transfer Learning in NLP with Transformer Models

- What is Transfer Learning?
  - Transfer learning involves pre-training a model on a large dataset and fine-tuning it for specific downstream tasks
  - Advantages
    - Reduces the need for task-specific labeled data
    - Speeds up training and improves performance on specialized tasks

Hands-On Exercise

- Objective
  - Experiment with a BERT variant (e.g., RoBERTa) and fine-tune it on an NLP task. Use the GPT-3 API for text generation and analyze the quality of the generated text

Day

07

Transformer Project – Text Summarization or Translation

Applying Transformer-Based Models to Advanced NLP Tasks

- Text Summarization
  - The process of condensing a piece of text while retaining the key information
  - Two types
    - Extractive Summarization: Selects key phrases or sentences from the original text
    - Abstractive Summarization: Generates new sentences that capture the meaning of the original text
- Text Translation
  - Converts text from one language to another while maintaining meaning and grammar
  - Examples
    - English to French translation
    - Multi-lingual translations with models like T5 or mT5

Fine-Tuning and Optimizing Models

- Pre-Trained Models for Summarization and Translation
  - T5 (Text-to-Text Transfer Transformer)
    - Treats every NLP problem as a text-to-text task
    - Fine-tuned for summarization and translation tasks
  - BART (Bidirectional and Auto-Regressive Transformer)
    - Combines BERT-like encoder and GPT-like decoder
    - Pre-trained for denoising and fine-tuned for summarization and translation
- Optimization Techniques
  - Learning rate scheduling
  - Hyperparameter tuning (batch size, optimizer type, maximum sequence length)

Analyzing Model Performance

- Evaluation Metrics
  - Text Summarization
    - ROUGE (Recall-Oriented Understudy for Gisting Evaluation)
    - BLEU (Bilingual Evaluation Understudy) for generated summaries
  - Text Translation
    - BLEU score for translation quality
    - Perplexity to measure model performance

Hands-On Project

- Objective
  - Fine-tune a pre-trained Transformer model (e.g., T5 or BART) for text summarization or translation and evaluate its performance

WEEK 13: Transfer Learning and Fine-Tuning

- Day 1 Introduction to Transfer Learning
- Day 2 Transfer Learning in Computer Vision
- Day 3 Fine-Tuning Techniques in Computer Vision
- Day 4 Transfer Learning in NLP
- Day 5 Fine-Tuning Techniques in NLP
- Day 6 Domain Adaptation and Transfer Learning Challenges
- Day 7 Transfer Learning Project – Fine-Tuning for a Custom Task

Day

01

Introduction to Transfer Learning

What is Transfer Learning?

- A machine learning technique where a model trained on one task is reused as a starting point for another related task
- Instead of training a model from scratch, pre-trained models are fine-tuned on a smaller dataset for a new task
- How It Differs from Traditional Training:

Aspect	Traditional Training	Transfer Learning
Starting Point	Train from scratch using random weights	Start with a pre-trained model
Training Time	Longer due to the need for learning basic features	Shorter as the model has already learned features
Dataset Size	Requires large datasets to perform well	Can work well on small datasets

Benefits of Transfer Learning

- Reduced Training Time
  - Pre-trained models already capture foundational features, so fewer epochs are needed
- Improved Performance on Small Datasets
  - Transfer learning allows effective training even when data is limited
- Leverages Generalization
  - Pre-trained models generalize better across tasks due to exposure to large-scale datasets

Applications of Transfer Learning

- In Computer Vision
  - Pre-trained models like ResNet, VGG, Inception, EfficientNet are used for
    - Object detection
    - Image classification
    - Image segmentation
- In NLP
  - Models like BERT, GPT, T5 are fine-tuned for
    - Text classification
    - Sentiment analysis
    - Named entity recognition
    - Question answering

Hands-On Exercise

- Objective
  - Set up a transfer learning environment, load a pre-trained model, and explore its architecture and layers

Day

02

Transfer Learning in Computer Vision

Popular Pre-Trained Models for Vision Tasks

- VGG
  - VGG16/VGG19: Deep networks with 16 or 19 layers
  - Known for simplicity in architecture: stack of convolutional layers followed by fully connected layers
  - Applications: General-purpose image classification and feature extraction
- ResNet
  - Residual Networks: Introduced residual connections (skip connections) to tackle vanishing gradients
  - Popular variants: ResNet18, ResNet50, ResNet101
  - Applications: Large-scale image classification tasks, object detection.
- Inception
  - InceptionV3: Known for inception modules, which allow for multi-scale feature extraction in one layer
  - Applications: Scene recognition, fine-grained image classification
- EfficientNet
  - Family of models that scales network depth, width, and resolution efficiently
  - Provides better performance with fewer parameters
  - Applications: Resource-constrained environments requiring high accuracy

Freezing and Unfreezing Layers for Fine-Tuning

- Why Freeze Layers?
  - Early layers in pre-trained models capture general features
  - Freezing these layers reduces training time and prevents overfitting on small datasets
- Why Unfreeze Layers?
  - Later layers learn task-specific features
  - Unfreezing layers allows the model to adapt to the new task
- Approach
  - Initial Training
    - Freeze most layers and train the last few layers
  - Fine-Tuning
    - Gradually unfreeze layers and reduce the learning rate for fine-tuning

Using Transfer Learning for Image Classification Tasks

- Steps
  - Load a pre-trained model (e.g., ResNet, VGG)
  - Replace the last layer with a task-specific classifier (e.g., softmax for multi-class classification)
  - Fine-tune the model on the new dataset

Hands-On Exercise

- Objective
  - Load a pre-trained ResNet or VGG model and fine-tune it for a new image classification task (e.g., classifying animals or plants)
  - Experiment with freezing and unfreezing layers and observe the impact on performance

Day

03

Fine-Tuning Techniques in Computer Vision

Choosing Layers to Fine-Tune and Understanding the Feature Extraction Process

- Feature Extraction in Pre-Trained Models
  - Early layers capture low-level features (e.g., edges, textures)
  - Middle layers capture mid-level features (e.g., shapes, parts of objects)
  - Late layers capture high-level features (e.g., specific objects, task-specific patterns)
- Choosing Layers to Fine-Tune
  - Freeze Early Layers: Retain general features learned during pre-training
  - Unfreeze Late Layers: Allow the model to adapt high-level features to the new task
- Best Practices
  - For small datasets: Fine-tune only the last few layers
  - For large datasets: Unfreeze more layers and fine-tune with a smaller learning rate

Data Augmentation for Improving Generalization

- What is Data Augmentation?
  - Artificially increase the diversity of training data by applying transformations like:
    - Rotation | Horizontal/vertical flipping | Scaling/zooming | Cropping | Color jittering
- Why Use Data Augmentation?
  - Reduces overfitting by introducing variability
  - Improves the model's ability to generalize to unseen data
- Examples of Augmentation
  - Rotation: Rotate images by random degrees (e.g., $\pm 15^{\circ}$)
  - Flip: Apply horizontal/vertical flips
  - Zoom: Randomly zoom in/out on images

Hyperparameter Tuning for Transfer Learning

- Key Hyperparameters
  - Learning Rate
    - A smaller learning rate is recommended for fine-tuning pre-trained models
    - Too large: Overshoots the optimal solution
    - Too small: Slow convergence
  - Batch Size
    - Larger batches stabilize training but require more memory
    - Smaller batches may lead to noisier updates but help with resource constraints
  - Optimizer
    - SGD: Works well with transfer learning when paired with momentum
    - Adam: Faster convergence but may require fine-tuning for stability
- Tuning Process
  - Start with default settings (e.g., learning rate: $1e-4$ 1e –4 , batch size: 32)
  - Experiment with one hyperparameter at a time to isolate its effect.

Hands-On Exercise

- Objective
  - Apply data augmentation to a dataset and train a fine-tuned model. Experiment with hyperparameters to observe their impact on performance

Day

04

Transfer Learning in NLP

Popular Pre-Trained NLP Models

- BERT (Bidirectional Encoder Representations from Transformers)
  - Architecture: Transformer-based encoder model
  - Training Tasks
    - Masked Language Modeling (MLM)
    - Next Sentence Prediction (NSP)
  - Applications
    - Text classification, sentiment analysis, question answering
- GPT (Generative Pretrained Transformer)
  - Architecture: Transformer-based decoder model
  - Training Task
    - Causal Language Modeling (predicting next word)
  - Applications
    - Text generation, summarization, dialogue systems

Popular Pre-Trained NLP Models

- T5 (Text-to-Text Transfer Transformer)
  - Treats all NLP tasks as text-to-text transformations
  - Applications
    - Summarization, translation, text classification
- RoBERTa (Robustly Optimized BERT)
  - Removes Next Sentence Prediction
  - Pre-trained on a larger dataset with optimized training strategies
  - Applications
    - Similar to BERT but with better performance on downstream tasks

Tokenization and Text Preprocessing for Fine-Tuning NLP Models

- Tokenization
  - Converts raw text into numerical representations
  - Types
    - WordPiece Tokenization: Used in BERT
    - Byte-Pair Encoding (BPE): Used in GPT and RoBERTa
- Text Preprocessing
  - Cleaning
    - Remove unnecessary characters (e.g., URLs, special symbols)
    - Normalization
      - Convert text to lowercase
      - Remove stopwords if necessary
    - Tokenization
      - Break text into tokens compatible with the pre-trained model

Adapting Pre-Trained Models for NLP Tasks

- Common Tasks
  - Text Classification
    - Categorize text into predefined labels (e.g., positive/negative sentiment)
  - Sentiment Analysis
    - Determine the sentiment polarity of text (e.g., positive, neutral, negative)
  - Summarization
    - Generate concise summaries from lengthy texts
- Steps
  - Load pre-trained model
  - Add a task-specific head (e.g., classification layer)
  - Fine-tune the model on task-specific data.

Hands-On Exercise

- Objective
  - Fine-tune a pre-trained BERT or T5 model for sentiment analysis using Hugging Face's Transformers library
  - Preprocess the text data, tokenize it, and evaluate the model

Day

05

Fine-Tuning Techniques in NLP

Fine-Tuning Methods for NLP Tasks

- Discriminative Fine-Tuning
  - Different layers of a pre-trained model capture different types of information
  - Approach
    - Use different learning rates for different layers of the model
    - Lower learning rates for early layers (general features)
    - Higher learning rates for later layers (task-specific features).
- Slanted Triangular Learning Rates (STLR)
  - Dynamically adjusts learning rates during training to balance exploration and convergence
  - Phases
    - Warm-Up: Gradually increase the learning rate to promote exploration
    - Decay: Slowly decrease the learning rate to ensure convergence
- Use Case
  - Effective for fine-tuning pre-trained models like BERT and GPT

Regularization and Dropout for Preventing Overfitting in NLP Models

- Regularization
  - L1 Regularization: Encourages sparsity by penalizing absolute weights
  - L2 Regularization (Ridge): Penalizes large weights to improve generalization
- Dropout
  - Randomly drops units (along with their connections) during training
  - Prevents over-reliance on specific neurons
  - Commonly used in Transformer-based models

Evaluating Model Performance with NLP-Specific Metrics

- Key Metrics
  - F1-Score
    - Harmonic mean of precision and recall
    - Suitable for classification tasks with imbalanced datasets
  - BLEU Score
    - Evaluates the quality of generated text against reference text
    - Commonly used for translation and summarization tasks
  - ROUGE Score
    - Measures overlap between generated and reference text
    - Used for summarization tasks

Hands-On Exercise

- Objective
  - Experiment with advanced fine-tuning techniques (e.g., STLR) on an NLP model and evaluate its performance using F1-score or BLEU score

Day

06

Domain Adaptation and Transfer Learning Challenges

Understanding Domain Adaptation and Handling Domain-Specific Data

- What is Domain Adaptation?
  - Domain adaptation involves transferring a model trained on one domain (source domain) to perform tasks in a different domain (target domain)
  - Example
    - Source domain: General news articles
    - Target domain: Medical text

Understanding Domain Adaptation and Handling Domain-Specific Data

- Why Domain Adaptation?
  - Many real-world tasks involve domain-specific data
  - Pre-trained models on general datasets might not perform optimally without adaptation
- Steps in Domain Adaptation
  - Fine-tune the pre-trained model on a domain-specific dataset
  - Incorporate domain-specific embeddings or vocabulary
  - Apply additional pre-training on the domain data if necessary

Challenges in Transfer Learning

- Data Mismatch
  - The source domain may not represent the target domain adequately
  - Example: General text (Wikipedia) vs. technical medical jargon
- Catastrophic Forgetting
  - During fine-tuning, the model may forget the knowledge learned from the source domain
- Computational Constraints
  - Fine-tuning large pre-trained models (e.g., BERT, GPT) requires significant computational resources

Strategies to Address Challenges

- Transfer Learning from Related Domains
  - Fine-tune on an intermediate domain dataset before adapting to the target domain
- Data Augmentation
  - Generate synthetic domain-specific data to augment the target dataset
  - Example
    - Use paraphrasing techniques to increase dataset diversity
- Domain-Specific Embeddings
  - Use pre-trained embeddings tailored to the target domain (e.g., BioBERT for biomedical data, LegalBERT for legal text)

Hands-On Exercise

- Objective
  - Fine-tune a pre-trained model on a domain-specific dataset (e.g., BERT for medical text classification) and experiment with domain-specific embeddings

Day

07

Transfer Learning Project
Fine-Tuning for a Custom Task

Applying Transfer Learning Techniques to a Custom Project

- Project Objective
  - Leverage transfer learning to solve a specific task in either computer vision or NLP
  - Fine-tune a pre-trained model for domain-specific data to achieve optimal performance
- Steps to Follow
  - Dataset Selection
    - Computer Vision: Custom image dataset (e.g., animal species classification)
    - NLP: Text classification task (e.g., sentiment analysis, product categorization)
  - Pre-Trained Model
    - Computer Vision: Models like ResNet, EfficientNet, or MobileNet
    - NLP: BERT, RoBERTa, or T5
  - Fine-Tuning Techniques
    - Regularization, hyperparameter tuning, data augmentation, discriminative learning rates

Analyzing Fine-Tuning Techniques

- Fine-Tuning Process
  - Freeze the pre-trained layers and train the custom classifier head first
  - Unfreeze some pre-trained layers for domain adaptation
  - Gradually reduce the learning rate to avoid catastrophic forgetting
- Key Techniques
  - Regularization
    - Dropout, L2 regularization to prevent overfitting
  - Data Augmentation
    - Enhance diversity in training data (rotation, cropping, or text paraphrasing)
  - Hyperparameter Tuning
    - Experiment with learning rate, batch size, and optimizer

Documenting Results and Baseline Comparisons

- Steps
  - Evaluate the baseline performance of the pre-trained model without fine-tuning
  - Track performance improvements after fine-tuning and hyperparameter optimization
  - Use metrics like accuracy, F1-score, BLEU, or ROC-AUC to compare results

Hands-On Exercise

- Objective
  - Fine-tune a pre-trained model (ResNet for computer vision or BERT for NLP) to solve a custom task
  - Evaluate and document results against a baseline

WEEK 14: Model Deployment and Serving

Day 1 Introduction to Model Deployment

Day 2 Creating REST APIs for Local Model Serving

Day 3 Containerizing Machine Learning Models with Docker

Day 4 Automating Model Deployment with Scripts

Day 5 Scaling Model Serving Locally

Day 6 Monitoring and Debugging Deployed Models Locally

Day 7 Local Deployment Capstone Project

Day

01

Introduction to Transfer Learning

Hands-On Project

- Objective
  - Fine-tune a pre-trained Transformer model (e.g., T5 or BART) for text summarization or translation and evaluate its performance

WEEK 15: Advanced Topics in Machine Learning Deployment

Day 1 Versioning Models for Continuous Deployment

Day 2 Securing Machine Learning APIs

Day 3 Optimizing Inference for Low Latency

Day 4 Handling Drift and Model Retraining

Day 5 Monitoring and Logging in Production

Day 6 Automating Workflows with Local CI/CD

Day 7 End-to-End Local Deployment System

WEEK 16: Real-Time ML and Advanced Serving Techniques

Day 1 Introduction to Real-Time Machine Learning

Day 2 Stream Processing for Machine Learning

Day 3 Optimizing Real-Time Inference

Day 4 Designing Real-Time Recommendation Systems

Day 5 Advanced Model Serving with Local Message Queues

Day 6 Edge Deployment for Real-Time Applications

Day 7 Real-Time ML Capstone Project