WEEK 1: Machine Learning System

1. What is Machine Learning (ML)?

- ML enables systems to **learn complex patterns from data** and make **predictions on unseen data**.
- Useful when:
  - Patterns are complex and repetitive.
  - Data exists or can be collected.
  - Predictions are needed at scale.
  - Wrong predictions are not costly.

When not to use ML:

- When simpler, cost-effective solutions work.
- If the use case is unethical.

2. ML Use Cases and Requirements

- **Examples**: Recommender systems, predictive typing, machine translation, fraud detection, smart health.
- **System Requirements**: Reliability, scalability, maintainability, adaptability.

3. ML Project Life Cycle

- **Stages**:
  Dataset → Data Pipeline → Feature Engineering → Model Training → Evaluation → Deployment → Monitoring.
- Development follows an **iterative and agile approach**.

4. ML Team Structure & Roles

- **Subject Matter Experts (SMEs)**: Define business questions/goals.
- **Data Scientists/ML Researchers**: Build and evaluate models.
- **Data Engineers**: Manage data pipelines (ETL).
- **ML Engineers**: Handle CI/CD, scalability, deployment, and monitoring.
- **ML Product Managers**: Bridge product needs with ML processes.

5. ML Systems vs Traditional Software

Traditional Software	ML Systems
Code and data are separated	Data and code are intertwined
Stable data dependencies	Data dependencies are unstable
Focus on code logic	Focus on data patterns

6. Research vs Production in ML

Research	Production
Focus on model performance	Focus on stakeholder needs, latency, fairness, interpretability
Static data	Constantly shifting data

7. Metrics and Problem Framing

- **Business Metrics**: Profit, sales, customer satisfaction.
- **ML Metrics**: Accuracy, F1-score, latency.
- Types of ML tasks: Regression, Binary Classification, Multiclass, Multilabel, Hierarchical Classification.

8. Key Features of MLOps

- Automates the ML lifecycle (CI/CD).
- Ensures collaboration, governance, monitoring, reproducibility.
- Mitigates risks like model degradation, fairness issues, or loss of skilled personnel.

9. Data and Feature Engineering

- Evaluate data availability, accuracy, and need for labeling.
- Perform feature engineering to improve model inputs.

10. Training, Evaluation & Deployment

- Track experiments and environment settings.
- Use reproducibility techniques.
- Deploy models via REST APIs, embedded models, or containerization (e.g., Docker).

11. Monitoring and Feedback Loop

- **Monitoring focus**:
  - **DevOps**: Performance, resource usage.
  - **Data Scientists**: Input drift, model accuracy over time.
  - **Business**: ROI, value delivery.
- Feedback mechanisms: Shadow testing, A/B testing.

12. Governance and Responsible AI

- Address **data governance**: data usage rights, PII protection, fairness.
- Ensure **process governance**: formalized steps for accountability.
- Incorporate **Responsible AI principles**:
  - Fairness, privacy, transparency, accountability.
  - Use tools like **Model Cards** for documenting model limitations and intended use.

13. Real-World Case Studies on Bias and Governance

- Examples: Recruitment model gender bias, automated grading system bias.

14. Discussion Exercise

- Case study: Developing an automated COVID-19 screening system.
- Focus areas: SME identification, business/ML metrics, data collection, responsible AI integration.

WEEK 2 : Machine Learning Fundamentals

1. What is Machine Learning (ML)?

- **Definition**: ML is a data analysis method that automates model building using algorithms that learn iteratively from data.
- **Purpose**: Allows computers to discover insights without explicit programming.

2. Why Machine Learning Matters

- **Automatic**: Once trained, models can operate autonomously.
- **Fast**: Handles big data faster than humans.
- **Accurate**: Improves prediction accuracy over manual methods.
- **Scalable**: Works effectively on large-scale datasets.

3. ML Development Workflow

1. **Data Acquisition**: Collecting data.
2. **Data Preparation**: Cleansing, shaping, enrichment, annotation (labeling).
3. **Model Training**: Training with training/validation/test sets.
4. **Model Testing**: Performance evaluation, optimization.
5. **Deployment & Monitoring**: Includes out-of-bag (OOB) testing, shadow deployment, and iteration.

4. Types of Machine Learning

Learning Type	Description	Example
Supervised Learning	Learning from labeled data	Classification, Regression
Unsupervised Learning	Learning from unlabeled data	Clustering, Anomaly Detection

Advanced ML	Includes CNN, Transformers, LLMs	Image recognition, Translation

5. Supervised Learning
- **Classification**: Predict categories (e.g., decision tree classifier).
- **Regression**: Predict continuous values (e.g., linear regression).
- **Examples**:
  - Gender prediction.
  - Loan customer lifetime value prediction.

6. Unsupervised Learning
- **Clustering**: Grouping similar data points.
- **Anomaly Detection**: Detecting outliers or abnormal cases.
- **Example Tools**:
  - Google Colab notebooks (links provided in the slides).
  - GitHub lecture materials (e.g., Decision Trees, Clustering, Anomaly Detection).

7. Common ML Issues
- **Overfitting**: Model performs well on training data but poorly on unseen data.
- **Unbalanced Data**: Disproportionate class distributions.
- **Noisy Data**: Unclean or irrelevant data.
- **Specialized vs. Generalized Learning**.
- **Optimization challenges**.

8. Foundations of Machine Learning
- **Mathematics & Statistics**: Regression, probability, Bayes theorem, entropy.
- **Computer Science**: Algorithms, data mining, HPC, optimization.
- **Artificial Intelligence**: Neural networks, reinforcement learning, computer vision, NLP.

9. Deployment and Operations
- **Deployment**: Moving ML models into production.
- **Production Environment**: Continuous delivery and integration.
- **Example Use Cases**:
  - Next Best Product Offering.
  - Loan Lifetime Value Prediction.

10. Advanced Concepts
- **Large Language Models (LLMs)**.
- **Generative AI**.
- **Transformers** for text translation (OpenAI Transformer architecture referenced).

WEEK 3 : Git and Version Control

- **Version Control Basics:**
  - Used for collaboration, tracking changes, and maintaining order in project files.
  - Two main types:
    - **Centralized Version Control (CVCS):** (e.g., CVS, Perforce, SVN) — developers pull from and push to a central server.
    - **Decentralized Version Control (DVCS):** (e.g., Git) — each developer has a full copy of the repository locally, allowing offline commits and branching.
- **Key Features of Good Version Control Systems:**
  - Store many versions.
  - Allow reverting to older versions.
  - Track the order of changes.
  - Manage reasonable version sizes.
  - Enable collaboration.
  - Help merge branched versions.
- **Git:**
  - Distributed version control system managing "commits" (snapshots) locally.
  - Tracks changes, allows reversion, and manages multi-developer work.
  - Handles merging (auto or manual).
- **Branching:**
  - Creates isolated "branches" for different work streams.
  - Commands include git checkout -b <branch>, git add, git commit, and switching branches with git checkout.
- **Pull Requests:**
  - Mechanism to safely propose and review changes before merging into the main branch.
  - Requires setting branch protection (e.g., requiring pull requests).
  - Pull requests are created after pushing changes from a new branch.
- **GitHub:**
  - Web-based Git repository hosting service (owned by Microsoft).
  - Features: repository hosting, pull requests, issue tracking, project wikis, CI/CD actions, and collaboration tools.
- **Lab Session:**
  - Practical work based on version control concepts.

To **push** your work to a new branch on GitHub, follow these steps:

✅ **1. Make sure you're on the right branch**

git checkout your-branch-name

✅ **2. Stage your changes**

git add .

✅ **3. Commit your changes**

git commit -m "Your commit message"

✅ **4. Push the branch to GitHub**

git push origin your-branch-name

WEEK 4 : Training Data

1. Importance of Training Data in ML
- **Core foundation** for supervised learning.
- Data quality significantly impacts model performance.
- Data availability challenges:
  - Scarcity in low-resource languages.
  - Domain-specific data difficulties (e.g., healthcare, finance).
  - Ground truth acquisition issues.
  - Sampling/selection bias and labeling bias.

2. Labeling Strategies

Hand Labeling
- Done by humans (subject matter experts or crowdsourcing platforms like Amazon Mechanical Turk).
- **Challenges**: Expensive, time-consuming, privacy concerns, slow iteration.
- **Label Multiplicity**: Different annotators may disagree.
  - Solution: Clear guidelines, training annotators, data lineage tracking.

Programmatic Labeling (Weak Supervision)
- Uses heuristics, keyword matching, regex, database lookups, and outputs of other models.
- Tools: **Snorkel**.
- **Advantages**:
  - Cost-effective.
  - Scalable.
  - Better privacy control.

Semi-Supervised Learning
- Combines small labeled datasets with large unlabeled datasets.
- Model predicts labels on unlabeled data; high-confidence predictions are reused for further training.

Transfer Learning
- Apply pre-trained models to new tasks with fine-tuning or prompt-based adjustments.

3. Class Imbalance Problem

- **Definition**: One class dominates the dataset (e.g., 99.99% normal vs. 0.01% cancer).
- **Challenges**:
  - Poor learning on rare classes.
  - Asymmetric error costs.
- **Solutions**:
  - Choosing proper evaluation metrics (Precision, Recall, F1-score).
  - Data-level approaches (e.g., data augmentation).
  - Algorithm-level approaches (e.g., cost-sensitive learning using cost matrices).

4. Choosing the Right Metrics

Accuracy	Not suitable for imbalanced data.
Precision	Important when false positives are costly (e.g., spam detection).
Recall	Important when false negatives are costly (e.g., cancer detection).
F1-score	Balances precision and recall; best for imbalanced datasets.

5. Data Augmentation Techniques

- **For NLP**: Synonym replacement (e.g., using GloVe embeddings).
- **For Computer Vision**: Image transformations (rotation, flipping, scaling).

6. Sampling Techniques for Data Selection

Method	Description	Pros/Cons
Random Sampling	Equal chance for each data point.	Rare events may be missed.
Stratified Sampling	Ensures subgroup representation.	Better for class balance.
Weighted Sampling	Higher chance for important samples.	Useful for emphasizing critical groups.
Non-Probabilistic Sampling	Convenience, snowball, judgment, quota sampling.	May introduce selection bias.

7. Handling Rare Events (e.g., Fraud Detection Case Study)

- If the fraud rate is low (0.2%), random sampling might miss fraud cases.
- Use **stratified** or **weighted sampling** to ensure coverage of rare but critical fraud patterns.

WEEK 5 : Containerization

1. The Problem with Traditional Software Deployment

- **Issues before containerization**:
  - Dependency conflicts (different software components need different library versions).
  - Environment inconsistencies (code works on one machine but not another).
  - Resource inefficiency when running multiple applications on the same machine.
  - Difficult scalability due to complex configurations.

2. Virtual Machines (VMs) as a Traditional Solution

- **How VMs work**:
  - Each VM runs its own full operating system.
  - High resource consumption (RAM, CPU, storage).
  - Slow startup (OS boot time).
  - Duplication of dependencies even when apps share libraries.
- **VM analogy**: Like building separate houses for each application.

3. Containerization Technology

- **What is a Container?**
  - A process that is isolated from the host system.
  - Packages all dependencies needed for an application.
  - Much lighter and faster than VMs (no full OS required).
- **Benefits**:
  - Portability across environments.
  - Reproducibility and reduced incompatibility issues.
  - Faster startup, more efficient resource use.
- **Popular Tool**: Docker (used throughout this course).
- **Container analogy**: Like apartments in the same building (shared foundation, isolated setup).

4. VMs vs. Docker (Comparison)

Feature	Virtual Machines	Docker Containers
OS Overhead	Full OS per VM	Shares host OS
Resource Usage	High (RAM, CPU, storage)	Low
Startup Time	Minutes (OS boot required)	Seconds
Scalability	Less efficient	Highly efficient
Portability	Limited to VM environment	Easy to run anywhere Docker is available

5. Docker Basics

Installation and Check

- Verify with: docker --version

Running Your First Docker Container

docker run hello-world

- Docker pulls the image from Docker Hub and runs a test script to confirm the setup.

6. Docker Desktop

- Docker Desktop provides a GUI interface to manage containers.

7. Shell Scripting with Docker

- Steps:
  - Fork the repo.
  - Clone it locally for your work.

8. Understanding the Dockerfile

What is a Dockerfile?

- A **Dockerfile** is a **text file** that contains a series of **instructions** on how to build a Docker image.
- It **automates** the creation of Docker images, ensuring consistency and reproducibility.
- Think of it like a **recipe**: each line tells Docker **how to assemble** the environment for your application.

How a Dockerfile Works

- When you run the command docker build, Docker reads the Dockerfile **line-by-line** and executes each instruction.
- Each instruction creates a **layer** in the final image.

- Layers make building faster by caching unchanged steps.

Common Dockerfile Instructions

Command	Purpose	Example
FROM	Set the base image	FROM python:3.10
COPY	Copy files from host to container	COPY . /app
RUN	Execute a command in the container	RUN pip install -r requirements.txt
CMD	Set the default command to run	CMD ["python", "app.py"]
EXPOSE	Inform Docker which port to listen on	EXPOSE 5000
WORKDIR	Set the working directory inside the container	WORKDIR /app
ENV	Set environment variables	ENV ENVIRONMENT= production

Simple Example of a Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:3.10

# Set the working directory
WORKDIR /app

# Copy the directory contents into the container
COPY . .

# Install dependencies
RUN pip install -r requirements.txt

# Expose port 5000
EXPOSE 5000

# Run app.py when the container launches
CMD ["python", "app.py"]
```

WEEK6: Building Data Pipeline (Airflow)

1. Overview of the Module

- Focus: **Apache Airflow**, building data pipelines, and error analysis.
- Goal: Understand how to automate, schedule, and monitor ML workflows effectively.

2. The Need for Workflow Management

- **Challenges without Workflow Management**:
  - Manual script scheduling with CRON lacks scalability and maintainability.
  - Difficult to manage hundreds of workflows, dependencies, and environment configurations.
  - Poor monitoring and troubleshooting of failed jobs.

3. Apache Airflow: A Solution for Workflow Management

- **What is Airflow?**
  - Open-source platform for authoring, scheduling, and monitoring workflows.
  - Defines workflows as **DAGs (Directed Acyclic Graphs)**.
  - Written in Python, supports distributed execution, and provides a friendly UI.
- **Airflow Features**:
  - Scalable, distributed task execution.
  - Workflow visualization (Grid View, Graph View, Calendar View, Gantt View).
  - Re-run specific steps, monitor failures, and track dependencies.

4. Core Concepts in Airflow (Next Column)

Concept	Description
DAG	Collection of tasks with dependencies, written as Python code.
Task	A node in the DAG representing a unit of work.
Operators	Define the type of work (e.g., PythonOperator, BashOperator, Sensor).
Hooks & Providers	Interface with external systems (e.g., MySQL, AWS, HDFS).
Connections	Store connection credentials securely via UI.

5. Types of Operators

Operator Type	Function
Action Operators	Execute a task (e.g., PythonOperator, BashOperator).
Transfer Operators	Move data between systems (e.g., MySqlToHiveOperator).
Sensor Operators	Wait for conditions (e.g., HttpSensor, SqlSensor).

6. Airflow Architecture and Workflow Execution

- **Scheduler** reads DAGs → creates DagRun → instantiates TaskInstances.
- **Executor/Workers** pull queued tasks → execute them → update task status (success, failed).
- **Web Server/UI** displays DAG status, task runs, logs, and execution history.

7. Monitoring and Visualization in Airflow

- **Grid View**: Task list with status.
- **Graph View**: Workflow graph with task relationships.
- **Calendar View**: Execution calendar.
- **Gantt View**: Task timing and overlap.
- **Logs**: Detailed execution logs per task.

8. Example Operators

Operator	Purpose
EmptyOperator	Placeholder, does nothing (used for grouping tasks).
BashOperator	Executes bash commands.
PythonOperator	Runs Python functions.
BranchPythonOperator	Enables conditional branching in workflows.
HttpSensor	Waits for HTTP response.

Operator	Purpose
SqlSensor	Waits for SQL conditions to be met.

WEEK7: Data Cleansing + Feature Engineering

Section 1: Data Cleansing (Data Wrangling)

1. Why Dirty Data Happens

- Causes:
  - Lack of input validation.
  - Software miscalculations or incomplete calculations.
  - Direct database manipulation.
  - Poor schema design.

2. Data Quality Dimensions

- **Accuracy, Validity, Reliability, Timeliness, Relevance, Completeness, Compliance**

3. Common Data Quality Problems

Problem Type	Single Source	Multi Source
Schema Level	Poor design, lack of integrity	Naming conflicts, structural issues
Instance Level	Misspellings, duplication	Inconsistent aggregating/timing

4. Topics in Data Cleansing

Problem Type	Example	Solution Approach
Missing Data	Missing departure time	Remove rows/columns, fill with zero or estimated values
Wrong Data	Negative ATM amounts	Remove or correct based on logic
Fragmented Data	Name split across fields	Merge into a common group
Outliers	Extremely high incomes	Clip or remove outliers based on analysis

Section 2: Feature Engineering

1. Common Techniques

Task	Purpose
Scaling	Normalize data for clustering/similarity analysis
Encoding Categorical Variables	Prepare non-numeric data for models
Dealing with Non-Linearity	Normalize skewed data distributions
Trend Calculation	Capture long-term and short-term patterns

2. Scaling Methods

Method	Range	Use Case
MinMaxScaler	0 to 1	Standard normalization
MaxAbsScaler	-1 to 1	Centered around zero

3. Encoding Categorical Variables

Encoding Type	Example
Binary Encoding	Yes/No → 1/0
Ordinal Encoding	Low, Medium, High → 0, 1, 2
Nominal Encoding	One-hot encoding (dummy variables)

4. Dealing with Non-Linearity

- Use **log transformation** or similar methods to handle skewed distributions or outliers.
- Ensure better visualization, clustering, or regression analysis.

5. Trend Analysis

Trend Type	Method
Long-term Trend	Slope from regression analysis
Short-term Trend	$T = M_0 - (M_1 + M_2)/2$, where M's are recent data points

1. Dealing with Missing Data

```
import pandas as pd

# Load data
df = pd.read_csv('your_data.csv')

# Check missing values
print(df.isnull().sum())

# Option 1: Drop rows with missing values
df_cleaned = df.dropna()

# Option 2: Fill missing values with 0
df_filled = df.fillna(0)

# Option 3: Fill missing values with mean (for numerical columns)
df['age'] = df['age'].fillna(df['age'].mean())
```

2. Handling Outright Wrong Data

```
# Detect negative values in 'amount' column
wrong_data = df[df['amount'] < 0]
print(wrong_data)

# Remove rows with wrong values
df = df[df['amount'] >= 0]
```

3. Fixing Fragmented Data

```
# Merge fragmented categorical fields

# Example: merging similar job titles
df['job'] = df['job'].replace({
    'software engineer': 'engineer',
    'developer': 'engineer',
    'senior engineer': 'engineer'
})
```

4. Handling Outliers

```
import numpy as np
import matplotlib.pyplot as plt
# Visualize distribution
plt.hist(df['income'], bins=50)
plt.show()
```

```
# Clip outliers (keep within 1st and 99th percentile)

lower_bound = np.percentile(df['income'], 1)
upper_bound = np.percentile(df['income'], 99)
df['income_clipped'] = np.clip(df['income'], lower_bound,
upper_bound)
```

Section 3: Coding

1. Scaling Features

```
from sklearn.preprocessing import MinMaxScaler, MaxAbsScaler

scaler = MinMaxScaler()  # or MaxAbsScaler()

# Assume numerical columns
numerical_cols = ['age', 'income', 'years_experience']

df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
```

2. Encoding Categorical Variables

```
# Binary Encoding
df['is_married'] = df['marital_status'].map({'Married': 1, 'Single': 0})

# Ordinal Encoding
education_mapping = {'High School': 1, 'Bachelor': 2, 'Master': 3,
'PhD': 4}

df['education_level'] = df['education'].map(education_mapping)

# One-Hot Encoding
df = pd.get_dummies(df, columns=['city'])
```

3. Dealing with Non-Linearity (Log Transformation)

```
# Log transform skewed data
df['log_income'] = np.log1p(df['income'])  # log(1 + income) to
handle zero values
```

4. Feature: Trend Calculation (for Time Series)

```
# Example: Short-term trend (simple version)
df['short_term_trend'] = df['sales_today'] - (df['sales_yesterday'] +
df['sales_day_before']) / 2
```

WEEK8: Model Parameter Tuning

2. Overfitting and Model Evaluation

- **Overfitting Case 1**: Testing on the training data gives misleadingly perfect results.
- **Overfitting Case 2**: Too complex models memorize training data, fail on unseen data.
- **Solution**: Use separate data for training and testing to evaluate model generalization.

Bias vs Variance Trade-off

- **Bias**: Error from assumptions in the model.
- **Variance**: Error from sensitivity to small fluctuations in the training set.

---

3. Hold-Out and Cross-Validation

- **Hold-Out Method**: Reserve part of data as test set (X_test, y_test).
- **Cross-Validation (CV)**:
  - Reduces overfitting risk on the test set.
  - **K-Fold CV**: Splits data into K parts, trains on K-1 parts, tests on the remaining part, rotates through all folds.

4. Hyperparameter Tuning

- **Hyperparameters**: Settings not learned from data (e.g., C, gamma in SVM, alpha in Lasso).
- Adjusted manually or via search to improve model performance.

5. Parameter Tuning Approaches

Method	Description
GridSearchCV	Exhaustively tries all parameter combinations.
RandomizedSearchCV	Samples random parameter combinations from a defined distribution.

6. Parameter Tuning Workflow

1. Define estimator (model).
2. Define parameter space.
3. Choose the search method (GridSearchCV / RandomizedSearchCV).
4. Apply cross-validation.
5. Evaluate using scoring metrics (e.g., accuracy, RMSE).
6. Use estimator.get_params() to view tunable parameters.

7. Nested Cross-Validation

- Combines two layers of cross-validation:
  - **Inner loop**: Parameter tuning.
  - **Outer loop**: Model evaluation.
- Avoids bias from selecting parameters based on the test set.

8. Example with Python (Scikit-Learn)

- **Loading data**, **splitting X and y**.
- Use of cross_validate, GridSearchCV, and evaluation metrics.
- Recommended practice: After tuning, validate the model with another unseen test set.

9. Evaluation Metrics

Metric	Purpose
Accuracy	Classification performance.
RMSE	Regression error measurement.

10. Example Use Case: Recommender System

- Tune hyperparameters to minimize RMSE.
- Select the algorithm with the best cross-validation performance

1. Basic Train/Test Split

```
from sklearn.model_selection import train_test_split

# Assume you have loaded your dataset
X = df.drop('target', axis=1)
y = df['target']

# Split into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

2. K-Fold Cross-Validation

```
from sklearn.model_selection import cross_validate
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

# Perform cross-validation
cv_results = cross_validate(model, X_train, y_train, cv=5,
return_train_score=True)

print(cv_results)
```

3. Hyperparameter Tuning with GridSearchCV

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# Define model
svc = SVC()

# Define parameter grid
```

```
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

# Setup GridSearchCV
grid_search = GridSearchCV(svc, param_grid, cv=5,
scoring='accuracy')

# Fit model
grid_search.fit(X_train, y_train)

# Best parameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Score:", grid_search.best_score_)
```

4. Hyperparameter Tuning with RandomizedSearchCV

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform

# Define model
svc = SVC()

# Define parameter distribution
param_dist = {
    'C': uniform(0.1, 10),
    'gamma': ['scale', 'auto'],
    'kernel': ['linear', 'rbf']
}

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(svc,
param_distributions=param_dist, n_iter=10, cv=5,
random_state=42)

# Fit model
random_search.fit(X_train, y_train)

# Best parameters and score
print("Best Parameters:", random_search.best_params_)
print("Best Cross-Validation Score:",
random_search.best_score_)
```

5. Evaluate the Best Model on the Test Set

```
from sklearn.metrics import accuracy_score

# Use the best model from grid search
best_model = grid_search.best_estimator_

# Predict and evaluate
y_pred = best_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)

print("Test Accuracy:", test_accuracy)
```

- Split → Cross-validate → Tune hyperparameters (GridSearchCV or RandomizedSearchCV) → Test best model.

WEEK9: Models and Experiment Tracking

2. Choosing an ML Model

- **Avoid the SOTA Trap**: SOTA models may not suit your use case due to resource, speed, or data differences.
- **Start Simple**: Use simpler models as baselines.
- **Avoid Bias in Evaluation**:
  - Equal experiment effort for all architectures.
  - Consider future scalability and data growth.
- **Balance Trade-offs**:
  - False positives vs false negatives.
  - Compute cost vs accuracy.
  - Interpretability vs accuracy.

3. Model Interpretability

- Importance of understanding **why** models make decisions.
- Use techniques like **Random Forest Feature Importance** to explain models.

4. Ensemble Methods Overview

Method	Description	Example
Bagging	Sample with replacement, aggregate predictions.	Random Forest
Boosting	Focus on misclassified samples in each iteration.	XGBoost, LightGBM
Stacking	Combine multiple models via a meta-model.	Model stacking (meta-learning)

5. Foundation Models and Transformers

- **Language Models (LMs)**: Probability distribution over token sequences.
- **Tokenizer Unfairness**: Tokenizer designs may introduce language biases.
- **Transformer Architecture**: Attention mechanism (key, query, value) enables contextual understanding.
- **Training Data Challenges**:
  - Underrepresentation of low-resource languages.
  - Consent and data privacy issues (e.g., Common Crawl concerns).

6. Training Compute Cost

- Large models require significant compute (e.g., GPT-3, Llama series).
- FLOPs and GPU requirements estimate cost and training time.
- Example: GPT-3 training could exceed $4 million at 70% utilization on NVIDIA H100 GPUs.

7. Experiment Tracking: Why and How

- **Purpose**:
  - Organize experiments systematically.
  - Enable reproducibility.
  - Track parameters, metrics, models, and metadata.

Experiment Tracking Tools

Tool	Key Features
MLflow	Experiment tracking, model registry, deployment.
Weights & Biases	Experiment monitoring and collaboration.

8. MLflow Essentials

- **Four Modules**:
  - Tracking (experiments, runs, metrics).
  - Models (save, load models).
  - Model Registry (manage production stages).
  - Projects (standardize reproducible code packaging).

UI Commands:

```
mlflow server --host 127.0.0.1 --port 8080
mlflow ui --backend-store-uri sqlite:///mlflow.db
```

- **Autologging Supported Libraries**: scikit-learn, XGBoost, LightGBM, PyTorch, etc.

WEEK10: Model Deployment

1. What is Model Deployment?

- The process of making a trained ML model available for **real-world use**.
- Allows:
  - **Real-time or batch predictions**.
  - Integration into business workflows.

- Scalability across multiple users and devices.

2. Types of Deployment

Type	Description	Tools/Platforms
Local Deployment	Offline testing or small-scale applications.	Local machines.
Cloud Deployment	Scalable, online applications.	AWS SageMaker, Google AI Platform, Azure ML.
Edge Deployment	For IoT/mobile devices; optimized models for limited hardware.	TensorFlow Lite, ONNX.
Containers & APIs	Package models using Docker and expose via Flask, FastAPI, TensorFlow Serving.	Docker, REST APIs.

3. Prediction Pipelines

Pipeline Type	Description	Example Scenario
Batch Prediction	Predict periodically, store results for later use.	Retail sales forecast done weekly.
Online Prediction	Predict in real-time as requests arrive.	Fraud detection on credit card transactions.

4. Real-World Use Cases

Scenario	Use of Prediction
Retail	Sales forecasting using batch prediction.
Banking	Fraud detection via real-time predictions.
Ride-sharing	Real-time matching of drivers and riders.

5. Model Optimization for Deployment (Inference Optimization Techniques)

Technique	Purpose	Tools/Examples
Quantization	Reduce model size by converting weights (float32 → int8/float16).	TensorRT, PyTorch Quantization, ONNX Runtime.
Pruning	Remove less important neurons/filters.	Trade-off: size vs. accuracy.
Distillation	Train a small model to mimic a large one.	Student-teacher model setup.

6. ONNX: Open Neural Network Exchange

- Universal format for deploying models across frameworks.
- Supports transferring models from **PyTorch, TensorFlow** to environments like **C++, mobile, web**.
- Enables efficient inference with engines like ONNX Runtime and TensorRT.

7. TensorRT for Efficient Inference

- NVIDIA's optimization toolkit for faster and more efficient inference.
- Supports quantization, pruning, and high-performance deployment.

8. Deployment with Docker

- Containerize your ML model and serve via REST APIs.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}|$$

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

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

Where,

$\hat{y}$ – *predicted value of* y

$\bar{y}$ – *mean value of* y

7. A classifier's confusion matrix for the classes "Normal" and "Disease" is shown below. (10 points)

		Predicted class		
	Class	Disease	Normal	Total
Actual class	Disease	270 =TP	280 = FN	550 = P
	Normal	2000 = FP	12000 = TN	14000 = N
	Total	2270	12280	14550 =ALL

7.1) What is the accuracy of this classifier? What is the accuracy of each class? (3 point)

ACCURACY = (270 +12000) / 14550 = 0.8433

7.2) What are the precision, recall and specificity of this classifier? (3 point)

SENSITIVTY = TP/ P = 270 / 550 = 0.4909,

SPECIFICITY = TN/ N = 12000 / 14000 = 0.8571

PRECISION = TP / (TP +FP) = 270/ (270 +2000) = 0.1189

RECALL = TP/ (TP+FN) = 270 /(270+280) = 0.4909

7.3) What is the F1-score of this classifier? (2 points)

F1-score = 2 (PRECISION)(RECALL) / (PRECISION + RECALL)
= 2 (0.1189)(0.4909) / (0.1189+0.4909) = 0.1914

7.4) Should we use this classifier with patients in the hospital? Why or why not? (2 points)

NO, because the sensitivity, specificity and f-measure are too low. The classifier is not reliable.

WEEK11: Model Monitoring

🧠 Why Model Monitoring Is Critical

After deployment, models can **fail silently** or degrade due to:

- **Software failures**: dependency, deployment, or hardware issues.
- **ML-specific failures**: input data shifts, label imbalance, or changing relationships between features and targets.

🔄 Types of ML-Specific Failures

Type	Description & Example
Covariate Shift	Input distribution **X** changes, but **P(Y

E.g., juice-buying behavior changes with age/climate.

Type	Description & Example
Label Shift	Output distribution **P(Y)** changes, but input-output relationship remains.

E.g., spam rate jumps from 5% → 30%.

Concept Shift	**P(Y
E.g., apartment price prediction changes post-earthquake.	

📊 Monitoring Techniques

Method	Description
Batch monitoring	Periodic jobs that evaluate models & store logs.
Real-time monitoring	Live metrics sent to monitoring service (e.g., Evidently, Prometheus).
Alerts	Notifications when metrics deviate from thresholds.
Dashboards	Visual summaries of model & data metrics.

📉 Beyond Standard Metrics

- **Accuracy, Precision, Recall** are useful but **not enough**.
- Production challenges include:
  - Delayed or no ground truth.
  - Business-specific priorities (e.g., churn rate, revenue).
- Use **custom & proxy metrics** tied to user behavior:
  - Clicks, watch time, purchases.
  - Revenue, cost, sentiment, churn.

📦 Data Quality Checks

- Schema validation, range checking, record volume monitoring.
- Ask: Does the data reflect reality? Is it complete, timely, and useful?

⚠️ Types of Data Drift

Type	Cause
Instantaneous	Deployment bugs or external shocks (e.g., COVID).
Gradual	Changes in user behavior over time.
Periodic	Seasonal trends, time zone effects.
Temporary	Short-term anomalies or attacks.

📐 Measuring Drift (KL Divergence Example)

```
from scipy.stats import entropy
import numpy as np

# Example: compare new and reference data distributions
Q = np.array([0.4, 0.3, 0.2, 0.1])  # training
P = np.array([0.1, 0.2, 0.3, 0.4])  # live
kl_div = entropy(P, Q)
print(f"KL Divergence = {kl_div:.4f}")
```

💻 System Metrics

Monitor resource usage to avoid system failures:

- mpstat, free -m, df -h, nvidia-smi

🔍 EvidentlyAI for Monitoring

- Open-source Python package for ML monitoring.
- Install: pip install evidently
- Features:
  - Model quality report
  - Data quality check
  - Drift detection

Drift options:

- all_features_stattest
- num_features_stattest
- cat_features_stattest
- per_feature_stattest