ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

◇ INTRODUCTION TO MODEL DEPLOYMENT: FLASK, FASTAPI

📌 INTRODUCTION

Machine Learning (ML) models are only valuable if they can be deployed and used in real-world applications. **Model deployment** is the process of making a trained ML model available as a **service** that other applications, websites, or users can interact with.

In this study material, we will explore how to deploy ML models using two popular Python web frameworks:

✓ **Flask** – A lightweight web framework for serving ML models.

✓ **FastAPI** – A modern, high-performance framework for fast model deployment.

📌 CHAPTER 1: UNDERSTANDING MODEL DEPLOYMENT

1.1 What is Model Deployment?

Model deployment involves **integrating a trained ML model into a production system** so that it can be accessed by users, applications, or APIs.

✓ **Training Phase:** Develop and train an ML model.

✓ **Deployment Phase:** Make the model available as an API or

service.

✓ **Inference Phase:** Users send input data, and the deployed model returns predictions.

🔹 **Example:**

A **fraud detection model** deployed as an API allows a banking system to send transaction details and receive fraud risk predictions in real-time.

1.2 Why Deploy an ML Model?

✅ **Enables real-time predictions** – Allows applications to use the model dynamically.

✅ **Makes ML models accessible** – Other systems can interact via APIs.

✅ **Allows for continuous updates** – The model can be retrained and redeployed.

✅ **Optimizes performance** – Helps scale the model for large users.

🔹 **Real-World Use Cases:**

- **Chatbots** use NLP models to process user queries.
- **Medical AI applications** analyze patient data via APIs.
- **Recommendation systems** personalize e-commerce experiences.

📌 Chapter 2: Deploying an ML Model with Flask

2.1 What is Flask?

Flask is a **lightweight, easy-to-use Python web framework** that allows developers to create APIs quickly. It is widely used for **small-scale ML model deployments**.

✓ Simple and flexible.
✓ Works well for small ML applications.
✓ Allows for easy API creation.

2.2 Steps to Deploy an ML Model using Flask

Step 1: Install Dependencies

pip install flask joblib pandas scikit-learn

Step 2: Load a Trained Model

```
import joblib
import numpy as np
from flask import Flask, request, jsonify

# Load trained model
model = joblib.load("model.pkl")

# Initialize Flask app
app = Flask(__name__)

@app.route('/predict', methods=['POST'])
def predict():
```

```
    data = request.get_json()
    input_data = np.array(data['features']).reshape(1, -1)
    prediction = model.predict(input_data)
    return jsonify({'prediction': int(prediction[0])})

if __name__ == '__main__':
    app.run(debug=True)
```

Step 3: Run the Flask API

python app.py

✓ **The API will be available at:** http://127.0.0.1:5000/predict

Step 4: Test the API Using Postman or cURL

curl -X POST http://127.0.0.1:5000/predict -H "Content-Type: application/json" -d '{"features": [5.1, 3.5, 1.4, 0.2]}'

✓ **The API will return:**

{"prediction": 0}

(assuming 0 represents the "Setosa" class in an Iris classification model).

📌 Chapter 3: Deploying an ML Model with FastAPI

3.1 What is FastAPI?

FastAPI is a **modern, high-performance web framework** for Python, designed specifically for APIs.

✓ **Faster than Flask** due to async support.
✓ **Automatic data validation** using Pydantic.
✓ **Built-in interactive API documentation.**

3.2 Steps to Deploy an ML Model using FastAPI

Step 1: Install Dependencies

```
pip install fastapi uvicorn joblib numpy scikit-learn pydantic
```

Step 2: Load a Trained Model & Create FastAPI App

```
import joblib
import numpy as np
from fastapi import FastAPI
from pydantic import BaseModel

# Load trained model
model = joblib.load("model.pkl")

# Initialize FastAPI app
app = FastAPI()

class ModelInput(BaseModel):
    features: list

@app.post('/predict')
```

```
def predict(data: ModelInput):
    input_data = np.array(data.features).reshape(1, -1)
    prediction = model.predict(input_data)
    return {"prediction": int(prediction[0])}
```

Step 3: Run the FastAPI Server

```
uvicorn app:app --host 0.0.0.0 --port 8000 --reload
```

✓ **The API will be available at:** http://127.0.0.1:8000/docs

Step 4: Test the API Using the Built-in Swagger UI

FastAPI provides **interactive API documentation** at http://127.0.0.1:8000/docs, where you can test API calls directly in the browser.

📌 Chapter 4: Comparing Flask vs FastAPI for Model Deployment

Feature	Flask	FastAPI
Performance	Slower	Faster (Async)
Data Validation	Manual	Built-in (Pydantic)
API Documentation	Manual	Auto-generated (/docs)
Best Use Case	Small-scale ML models	Large-scale, real-time ML APIs

✓ **Choose Flask** if you need a **quick and simple deployment**.

✓ **Choose FastAPI** if you need **high performance and scalable APIs**.

📌 Chapter 5: Exercises & Assignments

5.1 Multiple Choice Questions (MCQs)

1. **Which framework is better for high-performance ML model deployment?**
   - (a) Flask
   - (b) FastAPI ✅
   - (c) Django
   - (d) TensorFlow
2. **What is the primary advantage of FastAPI over Flask?**
   - (a) Faster execution ✅
   - (b) Less code
   - (c) No dependencies
   - (d) Built-in HTML support
3. **Which library is used to load trained ML models in Python?**
   - (a) Pandas
   - (b) NumPy
   - (c) Joblib ✅
   - (d) OpenCV

5.2 Practical Assignments

📌 **Task 1:** Deploy an **Iris classification model** using **Flask** and test it using Postman.

📌 **Task 2:** Deploy a **spam detection model** using **FastAPI** and test it in the browser.

📌 **Task 3:** Compare **response times** of Flask vs FastAPI for the same ML model.

📜 SUMMARY

✓ **Model Deployment** makes ML models accessible via APIs.

✓ **Flask** is **lightweight** and suitable for small-scale deployments.

✓ **FastAPI** is **faster, scalable,** and provides **automatic API documentation**.

✓ **Both frameworks enable real-time predictions** for applications.

◇ ML Model Optimization & Hyperparameter Tuning

📌 Introduction

Machine Learning models require **fine-tuning** to achieve optimal performance. **Hyperparameter tuning** is the process of adjusting **hyperparameters** (external parameters set before training) to **maximize model accuracy and efficiency**.

In this study material, we will cover:

✓ **The difference between parameters and hyperparameters**

✓ **Common hyperparameters in ML models**

✓ **Techniques for hyperparameter tuning**

✓ **Best practices for model optimization**

📌 Chapter 1: Understanding Hyperparameters & Model Optimization

1.1 What Are Hyperparameters?

Hyperparameters are **configurations that control the training process** of a machine learning model but are not learned from the data. They must be set manually before training.

🔹 **Examples of Hyperparameters:**

✓ **Learning rate (α)** – Controls how much weights update in each step.

✓ **Number of layers & neurons (Deep Learning models)** – Affects model complexity.

✓ **Batch size** – Number of samples processed before model updates.

✓ **Number of trees (Random Forest, XGBoost)** – Determines ensemble size.

1.2 Parameters vs. Hyperparameters

Feature	Parameters	Hyperparameters
Definition	Learned during training	Set before training
Examples	Weights, Biases	Learning rate, Batch size, Number of epochs
Adjusted By	The model	The user (or tuning algorithm)
Optimization	Done using backpropagation	Done using tuning methods

✓ **Example:**

- In a **linear regression model**, **parameters** include the slope and intercept.
- In **deep learning**, **hyperparameters** include **learning rate, number of layers, and activation functions**.

📌 Chapter 2: Common Hyperparameters in Machine Learning Models

2.1 Hyperparameters in Supervised Learning Models

🔹 **Linear Regression:**

✓ Regularization strength (**L1/L2 penalty**)

✓ Learning rate

- **Decision Trees:**

✓ Maximum depth of the tree

✓ Minimum samples per split

- **Random Forest:**

✓ Number of trees

✓ Maximum depth

- **Support Vector Machines (SVMs):**

✓ Kernel type (linear, polynomial, RBF)

✓ Regularization parameter (**C**)

- **Neural Networks:**

✓ Number of hidden layers & neurons

✓ Learning rate

✓ Batch size

✓ Dropout rate

📌 Chapter 3: Techniques for Hyperparameter Tuning

3.1 Grid Search

Grid Search is an **exhaustive search** technique where we specify a list of hyperparameter values and evaluate **every combination**.

✓ **Example:** Tuning learning rate & batch size in an **MLP model**.

```
from sklearn.model_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier


# Define hyperparameters

param_grid = {
```

```
    'n_estimators': [10, 50, 100],
    'max_depth': [5, 10, None]
}

# Initialize classifier
rf = RandomForestClassifier()

# Perform Grid Search
grid_search = GridSearchCV(rf, param_grid, cv=3,
scoring='accuracy')
grid_search.fit(X_train, y_train)

# Best parameters
print("Best Parameters:", grid_search.best_params_)
```

✓ **Pros:** Finds the best combination of parameters.
✓ **Cons:** Computationally expensive for large search spaces.

3.2 Random Search

Instead of evaluating all hyperparameter combinations, **Random Search** selects random sets of hyperparameters.

✓ **Example:**

```
from sklearn.model_selection import RandomizedSearchCV
import numpy as np
```

```
# Define hyperparameter grid
param_dist = {
   'n_estimators': np.arange(10, 200, 10),
   'max_depth': [5, 10, None]
}

# Perform Random Search
random_search = RandomizedSearchCV(rf, param_dist, n_iter=10, cv=3, scoring='accuracy')
random_search.fit(X_train, y_train)

# Best parameters
print("Best Parameters:", random_search.best_params_)
```

✓ **Pros:** Faster than Grid Search.
✓ **Cons:** Might miss the best combination.

3.3 Bayesian Optimization

Bayesian Optimization **intelligently selects the next set of hyperparameters** based on previous evaluations using probability models.

✓ **Example using Optuna:**

```
import optuna
```

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


# Define objective function
def objective(trial):
    n_estimators = trial.suggest_int("n_estimators", 10, 200, step=10)
    max_depth = trial.suggest_int("max_depth", 5, 20)


    model = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth)
    score = cross_val_score(model, X_train, y_train, cv=3, scoring='accuracy').mean()


    return score


# Run optimization
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=10)


# Best parameters
print("Best Parameters:", study.best_params_)
```

✓ **Pros:** Efficient, learns from previous iterations.

✓ **Cons:** More complex than Grid or Random Search.

📌 Chapter 4: Best Practices for Model Optimization

4.1 Cross-Validation

Cross-validation ensures the model generalizes well across unseen data.

✓ Example:

```
from sklearn.model_selection import cross_val_score

cv_score = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')

print(f"Cross-validation accuracy: {cv_score.mean():.2f}")
```

4.2 Regularization Techniques

Regularization **prevents overfitting** by penalizing complex models.

✓ Example (L1/L2 Regularization in Logistic Regression):

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(penalty='l2', C=0.1)
```

4.3 Early Stopping

Stops training when validation performance stops improving.

✓ Example in Keras:

```
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
```

📌 Chapter 5: Exercises & Assignments

5.1 Multiple Choice Questions (MCQs)

1. **Which hyperparameter controls the learning step size in gradient descent?**
   - (a) Batch size
   - (b) Regularization strength
   - (c) Learning rate ✅
   - (d) Dropout rate
2. **Which hyperparameter determines the maximum depth of a decision tree?**
   - (a) n_estimators
   - (b) max_depth ✅
   - (c) learning_rate
   - (d) kernel
3. **Which technique selects hyperparameters randomly from a given range?**
   - (a) Grid Search
   - (b) Random Search ✅
   - (c) Bayesian Optimization
   - (d) Cross-Validation

5.2 Practical Assignments

📌 **Task 1:** Tune hyperparameters for a **Random Forest model** using **Grid Search and Random Search**.

📌 **Task 2:** Implement **Bayesian Optimization** for an **XGBoost model**.

📌 **Task 3:** Apply **Early Stopping** in a **Neural Network** to prevent overfitting.

📜 Summary & Key Takeaways

✅ **Hyperparameters** are set before training, while **parameters** are learned during training.

✅ **Grid Search** exhaustively tests all combinations but is slow.

✅ **Random Search** is faster but may miss the best combination.

✅ **Bayesian Optimization** learns from previous trials for efficient tuning.

✅ **Regularization, Cross-Validation, and Early Stopping** help improve model performance.

◇ Deploying ML Models on Cloud (AWS, GCP, Azure)

📌 Introduction

Deploying machine learning (ML) models on the cloud enables businesses and developers to **scale models efficiently, reduce infrastructure costs**, and **serve AI-driven applications in real-time**. **Cloud platforms like AWS, Google Cloud (GCP), and Microsoft Azure** provide a variety of services for deploying, managing, and monitoring ML models.

Why Deploy ML Models on the Cloud?

✓ **Scalability** – Handle large workloads dynamically.
✓ **Cost-Effective** – Pay only for the resources used.
✓ **Accessibility** – Deploy and access models globally.
✓ **Automated Infrastructure** – No need for complex hardware setups.
✓ **Security & Reliability** – Cloud providers offer built-in encryption, authentication, and data protection.

This study material covers **cloud-based ML deployment**, including **deployment strategies, tools, and step-by-step guides for AWS, GCP, and Azure**.

📌 Chapter 1: Overview of Cloud ML Model Deployment

1.1 What is Cloud Deployment?

Cloud deployment refers to **hosting ML models on cloud-based infrastructure** so they can be accessed via an API, web app, or mobile app.

1.2 Key Components of ML Model Deployment

1. **Trained ML Model** – The final model obtained after training and evaluation.
2. **Cloud Infrastructure** – Computing resources to host and serve the model.
3. **API Endpoint** – A RESTful or gRPC API to interact with the model.
4. **Monitoring System** – Tools to track model performance, errors, and drift.

1.3 Deployment Strategies for ML Models

1. **Batch Processing** – The model processes data in batches (e.g., fraud detection).
2. **Real-Time Inference** – The model responds to API requests instantly (e.g., chatbots).
3. **Edge Deployment** – The model runs on local devices instead of the cloud (e.g., IoT applications).

📌 Chapter 2: Deploying ML Models on AWS (Amazon Web Services)

2.1 Introduction to AWS for ML Deployment

AWS provides a range of ML services, including **Amazon SageMaker**, **AWS Lambda**, and **EC2 for scalable deployment**.

2.2 Steps to Deploy ML Models on AWS SageMaker

1. **Prepare Model Artifacts**
    - Train the model and save it as a .pkl or .h5 file.

2. **Upload the Model to S3**
3. import boto3
4. s3 = boto3.client('s3')
5. s3.upload_file('model.pkl', 'your-bucket-name', 'model.pkl')
6. **Create a SageMaker Endpoint**
    - Use SageMaker's built-in algorithms or bring your own model.
7. **Deploy the Model as an API**
8. from sagemaker import Model
9. model = Model(model_data='s3://your-bucket-name/model.tar.gz', role='your-role')
10. predictor = model.deploy(instance_type='ml.m5.large')
11. **Make Predictions via API**
12. response = predictor.predict([[5.1, 3.5, 1.4, 0.2]])
13. print(response)

2.3 Alternative AWS Deployment Methods

- **AWS Lambda** – Serverless ML deployment for lightweight models.
- **AWS EC2** – Custom ML model deployment with complete server control.

📌 Chapter 3: Deploying ML Models on Google Cloud Platform (GCP)

3.1 Introduction to GCP AI Platform

Google Cloud provides **Vertex AI, Cloud Run,** and **Cloud Functions** for deploying ML models.

3.2 Steps to Deploy ML Models on Vertex AI

1. **Train and Save the Model**
2. import pickle
3. pickle.dump(model, open('model.pkl', 'wb'))
4. **Upload Model to Google Cloud Storage (GCS)**
5. gsutil cp model.pkl gs://your-bucket-name/
6. **Deploy Model on Vertex AI**
    - Register the model in Vertex AI.
    - Create an **endpoint** for serving predictions.
7. **Make API Requests**
8. import requests
9. url = "https://your-endpoint-url"
10. data = {"instances": [[5.1, 3.5, 1.4, 0.2]]}
11. response = requests.post(url, json=data)
12. print(response.json())

3.3 Alternative GCP Deployment Methods

- **Cloud Run** – Deploy models as serverless APIs.
- **Cloud Functions** – For event-driven ML applications.

📌 Chapter 4: Deploying ML Models on Microsoft Azure

4.1 Introduction to Azure Machine Learning

Azure provides **Azure ML Studio, AKS (Azure Kubernetes Service),** and **Azure Functions** for ML deployment.

4.2 Steps to Deploy ML Models on Azure ML Studio

1. **Register and Train Model in Azure ML Studio**
2. **Save and Deploy the Model to Azure ML Service**
3. from azureml.core import Workspace, Model
4. ws = Workspace.from_config()
5. model = Model.register(ws, model_name='my_model', model_path='model.pkl')
6. **Create an API Endpoint**
7. from azureml.core.webservice import AciWebservice
8. deployment_config = AciWebservice.deploy_configuration(cpu_cores=1, memory_gb=1)
9. **Deploy and Test Model**
10. service = Model.deploy(ws, "my-service", [model], deployment_config)
11. service.wait_for_deployment()
12. print(service.scoring_uri)

4.3 Alternative Azure Deployment Methods

- **Azure Kubernetes Service (AKS)** – For large-scale deployments.
- **Azure Functions** – Event-driven ML services.

📌 Chapter 5: Challenges in Cloud-Based ML Deployment

1. **Cost Management** – Managing cloud costs efficiently.
2. **Model Performance** – Ensuring models run optimally.
3. **Security & Compliance** – Protecting sensitive data.
4. **Model Updates** – Handling new versions without downtime.

📌 Chapter 6: Exercises & Assignments

6.1 Multiple Choice Questions (MCQs)

1. **Which AWS service is best for real-time ML model deployment?**
   - (a) AWS EC2
   - (b) AWS SageMaker ✅
   - (c) AWS S3
   - (d) AWS CloudTrail
2. **Which GCP service is used for deploying ML models?**
   - (a) Vertex AI ✅
   - (b) Cloud Storage
   - (c) Google Drive
   - (d) BigQuery
3. **What is the purpose of an API endpoint in ML model deployment?**
   - (a) To store training data
   - (b) To allow models to make predictions ✅

- (c) To monitor cloud costs
- (d) To encrypt ML models

6.2 Practical Assignments

- **Task 1:** Deploy a pre-trained TensorFlow model on AWS SageMaker.
- **Task 2:** Deploy a simple Flask API for ML inference on Google Cloud Run.
- **Task 3:** Research how **containerized ML models** are deployed using **Docker & Kubernetes**.

📜 Summary & Key Takeaways

- **Cloud ML deployment** enables **scalability, cost-effectiveness, and real-time AI applications**.
- **AWS SageMaker, GCP Vertex AI, and Azure ML Studio** provide streamlined ML deployment services.
- **Models can be deployed as API endpoints** for real-time inference.
- **Understanding different deployment strategies** (batch, real-time, edge) is crucial for success.
- **Monitoring, updating, and securing ML models** is essential for maintaining performance.

◇ ML Ethics & Bias in AI

📌 Introduction

As **Machine Learning (ML) and Artificial Intelligence (AI)** become an integral part of modern society, ethical considerations are more important than ever. AI systems influence decisions in **hiring, lending, healthcare, law enforcement, and social media**, making it crucial to ensure fairness, transparency, and accountability.

Key Ethical Concerns in AI:

✓ **Bias & Discrimination** – AI models can amplify societal biases.
✓ **Privacy Violations** – AI can misuse personal data.
✓ **Lack of Transparency** – Many AI decisions are difficult to interpret.
✓ **Accountability Issues** – Who is responsible when AI makes a mistake?

This study material explores the ethical challenges in AI, **how bias arises**, and ways to **mitigate bias and ensure fairness** in machine learning systems.

📌 Chapter 1: Understanding Ethics in AI

1.1 What is AI Ethics?

AI ethics refers to the **moral principles and guidelines** that govern how AI should be developed, deployed, and used responsibly. The goal is to ensure that AI **benefits humanity** while minimizing harm.

🔹 **Example:**
A **hiring algorithm** trained on past hiring data **rejects female applicants** because historical hiring was biased towards men. This highlights **bias in AI models** and raises ethical concerns.

1.2 The Importance of AI Ethics

✓ **Prevents discrimination in hiring, lending, and legal systems.**
✓ **Ensures AI is aligned with human values and rights.**
✓ **Builds trust between AI developers and users.**
✓ **Avoids potential legal consequences of unfair AI decisions.**

- **Real-World Example:**
  - In 2018, **Amazon's AI hiring tool** was found to **favor male candidates** over female ones, as it learned from biased historical data.

📌 Chapter 2: Bias in AI & Machine Learning

2.1 What is Bias in AI?

Bias in AI refers to **systematic errors** that cause unfair treatment of individuals or groups. Biases can arise **at different stages** of an AI model's lifecycle, including **data collection, training, and decision-making**.

- **Example:**

A **facial recognition system** is trained mostly on lighter-skinned individuals. When tested on darker-skinned individuals, it produces incorrect results, leading to discrimination.

2.2 Types of Bias in AI

1. **Data Bias:**
   - Occurs when training data is **not diverse** or **represents only a specific group**.

- **Example:** A chatbot trained on biased social media conversations learns **offensive language**.

2. **Algorithmic Bias:**

- AI models may **amplify** or **exaggerate existing patterns** in data.
- **Example:** A lending algorithm **denies loans** to low-income groups based on past trends.

3. **Confirmation Bias:**

- AI models reinforce **pre-existing human beliefs** instead of providing diverse perspectives.
- **Example:** Search engines show results that align with a user's past browsing history.

4. **Selection Bias:**

- When data is **not representative** of the real-world population.
- **Example:** A **medical AI trained only on male patients** fails to diagnose diseases accurately in women.

📌 Chapter 3: Case Studies on AI Bias

3.1 Case Study 1: Racial Bias in Facial Recognition

✓ AI facial recognition systems from **Amazon, IBM, and Microsoft** were found to have **higher error rates** when identifying **darker-skinned and female individuals**.
✓ The technology was used in **law enforcement,** leading to **false arrests and wrongful accusations**.

3.2 Case Study 2: Gender Bias in AI Hiring Tools

✓ **Amazon's AI hiring tool** was biased against **female candidates** because it was trained on **male-dominated hiring data**.
✓ The system **downgraded resumes with the word "women"** (e.g., "women's soccer club").

3.3 Case Study 3: Healthcare AI Discrimination

✓ **An AI healthcare system** allocated **fewer resources** to Black patients because historical data **undervalued** their medical needs.
✓ The AI model **prioritized wealthier patients, ignoring real medical urgency**.

📌 Chapter 4: Strategies to Reduce AI Bias

4.1 Collecting Fair & Balanced Data

✓ Ensure **datasets represent all demographic groups**.
✓ Regularly audit data to detect **imbalances** and **missing information**.
✓ **Use synthetic data** to correct biases in small datasets.

4.2 Algorithmic Fairness & Transparency

✓ Implement **Fairness Constraints** – Adjust AI models to **treat all groups equally**.
✓ Use **Explainable AI (XAI)** – Ensure AI decisions are **transparent and interpretable**.
✓ Conduct **bias impact assessments** before deploying AI systems.

4.3 Human Oversight in AI Decision-Making

✓ AI should assist decision-making, not **replace human judgment**.
✓ Include **diverse teams** when developing AI models.
✓ Implement **bias-detection tools** to monitor AI fairness.

4.4 Ethical AI Development Frameworks

Organizations have introduced **AI ethics guidelines** to promote responsible AI:

✓ **Google AI Principles** – AI should be **socially beneficial** and **avoid unfair bias**.
✓ **EU AI Regulation** – Mandates **transparency, accountability, and fairness**.
✓ **IEEE Ethics Guidelines** – Focuses on **human-centric AI design**.

📌 Chapter 5: Exercises & Assignments

5.1 Multiple Choice Questions (MCQs)

1. **What is a major cause of bias in AI?**
   - (a) Lack of training
   - (b) Biased data ✅
   - (c) AI transparency
   - (d) Model accuracy
2. **How can AI bias be minimized?**
   - (a) Using diverse training data ✅
   - (b) Ignoring AI audits

- (c) Removing fairness constraints
- (d) Keeping AI fully autonomous

3. **Which case study highlighted gender bias in hiring?**
   - (a) Amazon's AI hiring tool ✅
   - (b) IBM Watson
   - (c) Google's AlphaGo
   - (d) Tesla's Autopilot

5.2 Practical Assignments

📌 **Task 1:** Research **real-world AI bias incidents** and write a short report on how they were addressed.
📌 **Task 2:** Use a small dataset to detect **bias in AI predictions** (e.g., sentiment analysis of biased tweets).
📌 **Task 3:** Propose a **framework for ethical AI development** in a specific industry (e.g., banking, healthcare).

📜 Summary

✓ **AI Ethics** focuses on ensuring **fair, accountable, and transparent** AI systems.
✓ **Bias in AI** arises from **unbalanced data, flawed algorithms, and human prejudices**.
✓ **Real-world case studies** show AI bias in **facial recognition, hiring, and healthcare**.
✓ AI bias can be reduced through **fair data collection, algorithmic fairness, and human oversight**.

✓ **Ethical AI development** is crucial to ensuring **trustworthy AI systems** for the future.

📌 ASSIGNMENT 1:

☑ DEPLOY A MACHINE LEARNING MODEL AS A REST API USING FLASK.

📌 Assignment Solution 1: Deploy a Machine Learning Model as a REST API using Flask

🎯 Objective

The goal of this assignment is to **deploy a trained machine learning model** as a RESTful API using the **Flask** web framework. This will enable external applications to interact with your model over HTTP requests, facilitating seamless integration and accessibility.

By the end of this guide, you will:

- Understand how to **serialize and save** a trained machine learning model.
- Learn to **set up a Flask application** to serve the model.
- Create **API endpoints** to handle prediction requests.
- **Test** the deployed API to ensure it functions correctly.

🛠️ Step 1: Train and Save Your Machine Learning Model

Before deploying, ensure you have a trained machine learning model. For demonstration, we'll use a simple **Logistic Regression** model trained on the **Iris dataset**.

```
# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression

import joblib


# Load the Iris dataset

iris = load_iris()

X = iris.data

y = iris.target


# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Train a Logistic Regression model

model = LogisticRegression(max_iter=200)

model.fit(X_train, y_train)


# Save the trained model to a file

joblib.dump(model, 'iris_model.pkl')
```

Explanation:

- **Data Loading:** We load the Iris dataset, which contains features of iris flowers and their corresponding species.
- **Model Training:** A Logistic Regression model is trained to classify the species based on flower features.

- **Model Saving:** The trained model is saved as iris_model.pkl using joblib for later use in our Flask application.

🛠️ Step 2: Set Up the Flask Application

Create a new directory for your project and navigate into it:

```
mkdir ml_flask_api

cd ml_flask_api
```

Initialize a virtual environment and install Flask:

```
# Create a virtual environment

python3 -m venv venv


# Activate the virtual environment

# On Windows

venv\Scripts\activate


# On macOS/Linux

source venv/bin/activate


# Install Flask

pip install Flask
```

Explanation:

- **Virtual Environment:** Creating a virtual environment ensures that dependencies are managed separately from system-wide packages.
- **Flask Installation:** Flask is installed within this environment to handle web requests.

🛠️ Step 3: Create the Flask API

Within your project directory, create a file named app.py:

```
# app.py


from flask import Flask, request, jsonify

import joblib

import numpy as np


# Initialize the Flask application

app = Flask(__name__)


# Load the trained model

model = joblib.load('iris_model.pkl')


# Define a route for the default URL

@app.route('/')

def home():
```

```
    return "Welcome to the Iris Prediction API!"


# Define a route for prediction
@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json(force=True)
    features = np.array(data['features']).reshape(1, -1)
    prediction = model.predict(features)
    return jsonify({'prediction': int(prediction[0])})


if __name__ == '__main__':
    app.run(debug=True)
```

Explanation:

- **Flask Initialization:** The Flask app is initialized.
- **Model Loading:** The saved Logistic Regression model is loaded using joblib.
- **Routes Defined:**
  - '/': A welcome message to confirm the API is running.
  - '/predict': Accepts POST requests with JSON data containing flower features and returns the predicted species.

🛠️ Step 4: Test the API Locally

Run the Flask application:

python app.py

Testing with curl:

In a separate terminal, you can test the API using curl:

```
curl -X POST -H "Content-Type: application/json" \
  -d '{"features": [5.1, 3.5, 1.4, 0.2]}' \
  http://127.0.0.1:5000/predict
```

Expected Response:

```
{
 "prediction": 0
}
```

Explanation:

- **Running the App:** The Flask app runs on http://127.0.0.1:5000/.
- **Testing:** We send a POST request with sample features to the /predict endpoint and receive a JSON response with the predicted class.

🛠 Step 5: Deploying to a Production Environment

For production deployment, consider using a production-ready server like **Gunicorn** and possibly containerizing your application with **Docker**.

Using Gunicorn:

Install Gunicorn:

pip install gunicorn

Run the application with Gunicorn:

gunicorn --bind 0.0.0.0:8000 app:app

Explanation:

- **Gunicorn:** A production-grade WSGI server that serves the Flask application efficiently.

Containerizing with Docker:

Create a Dockerfile:

```
# Use an official Python runtime as a parent image

FROM python:3.8-slim


# Set the working directory

WORKDIR /app


# Copy the current directory contents into the container

COPY . /app


# Install the required packages

RUN pip install --no-cache-dir -r requirements.txt


# Expose port 8000 for the API

EXPOSE 8000
```

```
# Define the command to run the application

CMD ["gunicorn", "--bind", "0.0.0.0:8000", "app:app"]
```

Explanation:

- **Dockerfile:** Defines the environment and instructions to containerize the Flask application, ensuring consistency across different deployment platforms.

📜 SUMMARY

In this assignment, we've:

- **Trained and saved** a machine learning model.
- **Set up a Flask application** to serve the model as a RESTful API.
- **Created endpoints** to handle prediction requests.
- **Tested the API locally** to ensure functionality.
- Discussed **production deployment** considerations using Gunicorn and Docker.

📌 ASSIGNMENT 2:

☑ OPTIMIZE A MODEL USING GRID SEARCH & HYPERPARAMETER TUNING.

📌 Assignment Solution 2: Optimize a Model using Grid Search & Hyperparameter Tuning

🎯 Objective

The goal of this assignment is to **optimize a machine learning model** by fine-tuning its hyperparameters using **Grid Search**. Hyperparameter tuning enhances model performance by systematically searching for the best parameter combinations.

By the end of this guide, you will:

- Understand the importance of hyperparameter tuning.
- Learn how to implement Grid Search using GridSearchCV from scikit-learn.
- Evaluate and select the best model based on optimized hyperparameters.

🛠 Step 1: Import Required Libraries

Begin by importing the necessary libraries.

```
# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score
```

Explanation:

- numpy and pandas are used for data manipulation.
- train_test_split and GridSearchCV assist in model evaluation and hyperparameter tuning.
- RandomForestClassifier is the machine learning model we'll optimize.
- accuracy_score measures the model's performance.

🛠 Step 2: Load and Prepare the Dataset

For this example, we'll use the **Iris dataset**, a classic dataset in machine learning.

```
# Load the Iris dataset

from sklearn.datasets import load_iris

data = load_iris()

X = pd.DataFrame(data.data, columns=data.feature_names)

y = pd.Series(data.target)


# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Explanation:

- The Iris dataset is loaded and split into features (X) and target (y).

- Data is divided into training (80%) and testing (20%) sets to evaluate model performance.

🛠️ Step 3: Define the Model and Hyperparameter Grid

We'll define a **Random Forest Classifier** and specify the hyperparameters to tune.

```
# Initialize the Random Forest Classifier
rf = RandomForestClassifier(random_state=42)


# Define the hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 200],      # Number of trees in the forest
    'max_depth': [None, 10, 20, 30],     # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],     # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],       # Minimum number of samples required to be at a leaf node
    'max_features': ['auto', 'sqrt', 'log2']  # Number of features to consider when looking for the best split
}
```

Explanation:

- RandomForestClassifier is chosen for its robustness and versatility.
- param_grid contains hyperparameters and their possible values to explore during Grid Search.

🛠 Step 4: Implement Grid Search with Cross-Validation

Use GridSearchCV to perform an exhaustive search over the hyperparameter grid.

```
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           cv=5, n_jobs=-1, verbose=2, scoring='accuracy')

# Fit GridSearchCV to the training data
grid_search.fit(X_train, y_train)
```

Explanation:

- cv=5 specifies 5-fold cross-validation.
- n_jobs=-1 utilizes all available processors for parallel computation.
- verbose=2 provides detailed logs of the process.
- scoring='accuracy' evaluates models based on accuracy.

🛠 Step 5: Evaluate the Best Model

After completing the Grid Search, identify and evaluate the best model.

```
# Retrieve the best parameters and estimator
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
```

```
# Predict on the test set
y_pred = best_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

print(f"Best Hyperparameters: {best_params}")
print(f"Test Set Accuracy: {accuracy:.2f}")
```

Explanation:

- best_params_ provides the optimal hyperparameter combination.
- best_estimator_ returns the model trained with the best parameters.
- Model performance is evaluated on the test set using accuracy.

📜 Summary

In this assignment, we've:

- **Loaded and prepared** the Iris dataset.
- **Defined a Random Forest model** and specified a hyperparameter grid.
- **Implemented Grid Search with Cross-Validation** to find the optimal hyperparameters.
- **Evaluated the optimized model** on a test set.

Key Takeaways:

- **Hyperparameter tuning** is crucial for enhancing model performance.
- **Grid Search** systematically explores combinations but can be computationally intensive.
- **Cross-Validation** ensures the model generalizes well to unseen data.