Name of Learners:- Shailaja Ramteke

Batch No:- B310

IT Vedant Institute of Bangalore

Internship Report Task 7

==

Objective: Make the model interpretable and deployable.
Tasks: ○ Use SHAP or LIME to explain model predictions. ○ Deploy the model locally or on cloud platforms like AWS/GCP using Flask/Django.
Deliverables: ○ A deployed model endpoint. ○ Documentation on model interpretability and deployment steps.

Week 7

Deliverables: -

1. Model Explainability Report:- SHAP or LIME Analysis: Visualizations and explanations of how individual features influence the model's predictions. Examples include SHAP summary plots, dependency plots, or LIME visualizations for specific instances. Case studies or examples of how the model's predictions align with domain knowledge or expected behaviour. Key Insights: Identification of the most influential features and their impact on predictions. Analysis of whether the model is making logical, interpretable predictions or requires further refinement.

2. Model Deployment Package:- Local Deployment (if applicable): A Flask/Django-based application that allows users to interact with the model via a simple API or web interface. Example endpoints for tasks such as making predictions (/predict) or getting feature explanations (/explain). Documentation for setting up and running the application locally. Cloud Deployment (if applicable): Deployment of the model on platforms like AWS, GCP, or Azure, including Server details (e.g., EC2 instance, App Engine configuration). Model endpoint URL for accessing predictions. Step-by-step instructions or scripts for deploying the application on the cloud platform.

3. User Interface/Experience Deliverables: - A functional interface for end-users to upload data and receive predictions and explanations. Documentation for using the application, including sample inputs and expected outputs.

4. Deployment Documentation: - Architecture Overview: High-level explanation of how the model is deployed and how users interact with it (e.g., user → API → model → response). Setup Instructions: Detailed steps for reproducing the deployment, including environment setup, dependencies, and configuration files. Monitoring and Maintenance Plan: Suggestions for monitoring the deployed model's performance and handling updates.

5. Reproducibility and Portability: - A packaged deployment script or containerized application (e.g., Docker file) for easy replication and portability.

Report

Report Juyperte nootbook link :- https://github.com/Shaila92/InternshipTask7

Objective: - The primary objective of this task is to:

- Make the machine learning model interpretable and understandable, enabling stakeholders to trust and validate its predictions.
- Deploy the trained model as a service, so that predictions can be accessed through an API, either locally or on a cloud platform (e.g., AWS, GCP).

Deliverable: -

1)Model Explainability Report:-

SHAP Analysis:- We utilized SHAP (SHapley Additive exPlanations) to interpret our trained Random Forest model on the phishing dataset. The SHAP visualizations include:

SHAP Summary Bar Plot:-Highlights the average magnitude of feature impacts on the model's output.

SHAP Summary Dot Plot:- Displays both the magnitude and direction of impact of each feature across samples.

SHAP Dependence Plot:- Demonstrates the relationship between a key feature's value and its SHAP value.

Case Study Example:- For selected instances from the test set, SHAP values illustrated how features like length_url, has_ip, and num_special_chars contributed to a phishing or not-phishing prediction. The results were consistent with domain expectations.

Key Insights:- The most influential features identified were length_url, has_ip, num_special_chars.
The model generally made logical, interpretable predictions.
No major anomalies or inconsistencies were observed that would warrant immediate model revision.

2) **Model Deployment Package :-**

Objective:-Deploy the trained phishing detection model as an API so users can interact with it either locally or through cloud platforms.

Local Deployment (Flask Application)

App Structure

- app.py — Flask application file.
- phishing_model.pkl — The trained and saved model.
- requirements.txt — Lists dependencies (Flask, NumPy, scikit-learn).

Example API Endpoints

- / → Health check endpoint (returns confirmation that API is running).
- /predict → POST request with JSON input for model prediction.

How to Set Up Locally:-Ensure Python is installed (Python 3.x).

Install dependencies:- pip install flask numpy scikit-learn

Run the Flask app:- python app.py

Test API using Postman or curl:

```
curl -X POST http://127.0.0.1:5000/predict -H "Content-Type: application/json" -d '{"features":
[0.5, 1, 0, 3, 2, 1, ...]}'
```

Cloud Deployment (Optional Example)

If deploying to AWS EC2 / GCP / Azure:

- Provision a virtual server (e.g., EC2 Ubuntu instance).
- Transfer your Flask app files using scp or git.
- Set up a virtual environment and install requirements.
- Use gunicorn + nginx (or similar) for production server.
- Open necessary ports (e.g., 5000 or 80) in the cloud firewall settings.

Example Cloud Endpoint

http://YOUR_SERVER_IP:5000/predict

Example Flask Code

```
from flask import Flask, request, jsonify

import pickle

import numpy as np

# Load the model

with open("phishing_model.pkl", "rb") as f:

    model = pickle.load(f)

app = Flask(__name__)

@app.route('/')

def home():

    return "Phishing URL Detection API is running!"

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

def predict():

    data = request.get_json(force=True)

    features = data.get('features')

    if features is None:

        return jsonify({"error": "No features provided"}), 400
    features_array = np.array(features).reshape(1, -1)
    prediction = model.predict(features_array)[0]
    return jsonify({"prediction": int(prediction)})


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

Example requirements.txt

```
flask
numpy
scikit-learn
```

Ensure that the input features match the model's expected feature count and order.

For production use, consider security, input validation, and rate-limiting.

For cloud deployments, SSL (HTTPS) should be added.

Deliverables:- Local Flask app with /predict endpoint.
Example code and documentation.
Guidance for optional cloud deployment.

3)User Interface/Experience Deliverables

Objective: Design and deliver a user-friendly interface that allows end-users to upload data, receive predictions, and view model explanations.

Components: Functional Interface:

- Framework: Flask-based web app.
- Features: Upload CSV or enter URL, Get prediction result: phishing or legitimate and Display SHAP-based explanation of the prediction.
- Endpoints:1) /: Home page with upload form. 2) /predict: Processes data and returns result. 3) /explain: Shows SHAP plots for uploaded data.

Example Flask Code (Jupyter-ready):

```
from flask import Flask, request, render_template_string
import pickle
import pandas as pd
import shap


# Load model
```

```
model = pickle.load(open("phishing_model.pkl", "rb"))


# Initialize Flask

app = Flask(__name__)


# Home page

@app.route("/")

def home():

  return '''

  <h1>Phishing Detection App</h1>

  <form action="/predict" method="post" enctype="multipart/form-data">

    Upload CSV file: <input type="file" name="file"><br><br>

    <input type="submit" value="Predict">

  </form>

  '''


# Predict route

@app.route("/predict", methods=["POST"])

def predict():

  file = request.files['file']

  df = pd.read_csv(file)

  preds = model.predict(df)

  return f"Predictions: {preds.tolist()}"


if __name__ == "__main__":

  app.run(debug=True)
```

User Documentation:How to Use:
Open the app in browser (default: http://127.0.0.1:5000/). Upload a CSV file with the same columns as used during training (without target). Click Predict to see the model output. (Optional) Extend app to show SHAP explanations visually.

- **Expected Input:** CSV file with feature columns like length_url, nb_dots, etc.
- **Expected Output:** JSON-like list of predictions (0 = legitimate, 1 = phishing).

Note:- SHAP explanation route can be added by generating plots and rendering them as images or embedding in HTML. Can extend using HTML templates, Bootstrap for styling, or JS for dynamic updates. Cloud deployment can reuse this interface with minor tweaks.

4)Deployment Documentation for Phishing Detection Model

Architecture Overview:- The deployed architecture consists of the following components:

- User Interface / Client: A web browser or API client sends HTTP requests (e.g., for predictions or explanations).
- Web Server: Flask-based API that handles incoming requests.
- Model Backend: The trained Random Forest model (phishing_model.pkl) loaded into memory for real-time inference.
- Response: The API processes input data, generates predictions or explanations, and returns results to the user.

Flow:-User → Flask API → Model (Random Forest Classifier) → Response → User

Setup Instructions:- Environment Setup

- Install Python 3.8+.
- Set up a virtual environment (recommended):
- python -m venv venv
- source venv/bin/activate # Linux/Mac
- venv\Scripts\activate   # Windows
- Install dependencies:
- pip install flask pandas scikit-learn shap lime

Directory Structure

```
project_root/
app.py              # Flask application
phishing_model.pkl      # Trained model file
templates/
index.html         # (Optional) UI template
 static/              # (Optional) CSS/JS
 requirements.txt       # Dependency list
```

Run the Application Locally

python app.py

Open http://127.0.0.1:5000 in a browser or API client.

Cloud Deployment (optional)

- AWS EC2 / GCP Compute / Azure VM:
    - Provision a server.
    - SSH into the server and clone your project.
    - Set up environment and run Flask app.
    - Use gunicorn + nginx for production-grade deployment.
- Platform-as-a-Service (e.g., Heroku / App Engine)
    - Add Procfile, runtime.txt, and deploy via CLI.

Monitoring & Maintenance Plan

- Logging: Log API requests and responses for auditing and debugging.
- Performance Monitoring:
    - Track latency and error rates (e.g., via AWS CloudWatch, GCP Stackdriver, or open-source tools like Prometheus).
- Model Drift:
    - Periodically evaluate model accuracy on fresh data.
    - Retrain model if performance degrades.
- Version Control:
    - Maintain versioned models (e.g., phishing_model_v1.pkl, phishing_model_v2.pkl).
- Security:
    - Implement API authentication (e.g., API keys).
    - Regularly update dependencies to address security vulnerabilities.

5) Phishing Detection Model Deployment Documentation:- **Reproducibility and Portability**

Packaged Deployment Script

We provide a deployment script (deploy.sh) that automates the following steps:

- Sets up a Python virtual environment.
- Installs required dependencies (requirements.txt).
- Launches the Flask application (app.py).

Example deploy.sh

```
#!/bin/bash

python3 -m venv venv

source venv/bin/activate

pip install -r requirements.txt

python app.py
```

Docker Containerization

We offer a Dockerfile to containerize the application for easy replication across different environments.

Example Dockerfile

```
FROM python:3.9-slim

WORKDIR /app

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

COPY . .

EXPOSE 5000


CMD ["python", "app.py"]
```

Usage Instructions

```
docker build -t phishing-detector .
```

```
docker run -p 5000:5000 phishing-detector
```

Notes:- Ensure Docker is installed and running. Adjust port mapping as needed. For cloud deployment, use container orchestration services (e.g., AWS ECS, GCP Cloud Run).

Overall Conclusion:-

The project successfully achieved its objective of making the phishing detection model interpretable, deployable, and reproducible. Below is a summary of the key outcomes from all five tasks:

1. Model Explainability Report Applied SHAP to visualize and interpret feature influences on model predictions. Identified important features like length_url, nb_dots, and domain_age that significantly impact phishing detection. Confirmed the model's decisions are logical and align with domain knowledge.
2. Model Deployment Package Developed a Flask API with endpoints for predictions and explanations. Provided setup documentation for local deployment. Explained deployment options for AWS/GCP.

3)User Interface / Experience Built a basic web interface for submitting URLs or features for prediction. Documented app usage with sample inputs and expected outputs.

4)Deployment Documentation Described architecture: user → API → model → response. Included detailed setup instructions and environment configuration. Proposed monitoring and maintenance strategies.

5)Reproducibility and Portability Provided a deployment script (deploy.sh) for quick setup. Designed a Dockerfile for containerized deployment to ensure portability. Shared clear build/run instructions for Docker.

Final Remark: The phishing detection system is production-ready, combining interpretability, ease of deployment, user-friendliness, and portability. The solution supports scaling and future improvements.