Machine Learning Project Documentation

Deployment

Group Member:

1. **Samiullah Gulzar**
2. **Hashmatullah Asady**
3. **Huma Akrami**
4. **Baiqra Muradi**

1. Overview

The deployment phase involves making the trained machine learning model available for real-world use. This process includes model serialization, serving, API integration, security measures, and monitoring to ensure reliable performance in a production environment. The model will be deployed as both an API using Flask and a web application using the Streamlit library.

Key Steps in Deployment:

1. **Getting the Model Ready**: We optimized the Random Forest model by saving it in an efficient format using `joblib` for quick loading and prediction.
2. **Choosing the Right Setup**: Based on the project's requirements, we selected a cloud-based deployment approach to ensure scalability and accessibility for healthcare providers.
3. **Connecting the Model**: An API was developed using Flask to allow seamless interaction with the model, enabling users to submit input data and receive predictions in real-time.
4. **Testing in the Real World**: The model was tested with simulated real-world data to validate its performance and ensure it could handle variations in input while maintaining accuracy.
5. **Keeping an Eye on Things**: This system requires continuous monitoring to ensure optimal performance and adaptability. By collecting new data and logs from incoming requests, we can identify patterns, detect potential issues, and gather insights to further improve the model over time. Advanced monitoring tools, like Grafana or Prometheus, are planned for implementation to provide real-time dashboards and alerts.

Why Deployment Matters:

Deploying this model ensures that it can deliver practical value by aiding early detection of respiratory infections in resource-limited settings. A well-executed deployment process bridges the gap between development and real-world impact, providing healthcare workers with an accessible and reliable tool to improve patient outcomes.

2. Model Serialization

Model serialization is a critical step in preparing the trained machine learning model for deployment. The trained model is serialized using the joblib library, which efficiently stores the model while preserving its structure and parameters. The model is saved in a `.pkl` format to allow quick loading during inference. we focused on effectively saving and loading the Random Forest model to ensure its seamless integration into the production environment.

Serialization Process

1. **Choosing the Serialization Format:** We used **joblib** for saving the model due to its efficiency in handling large objects like Random Forests. Joblib is particularly effective in managing NumPy arrays and complex data structures.
2. **Saving the Trained Model:** After training, the model was serialized and stored in a binary format to ensure quick loading during deployment:

```
[18]: import joblib
      joblib.dump(rf_model, "model.pkl")  # Save the trained model

[18]: ['model.pkl']
```

3. **Loading the Model:** During deployment, the serialized model is loaded back into memory for making predictions:

```
[19]: from fastapi import FastAPI
      from pydantic import BaseModel
      # import joblib
      # import numpy as np

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

4. **Preprocessing Consistency:** Preprocessing steps, such as feature scaling or encoding categorical variables, were stored alongside the model to maintain consistency between training and production data.

Considerations for Efficient Storage

1. **Compression:** For large models, we used compression to reduce file size while maintaining performance. Example:

```
1 # Saved model with compression
2 joblib.dump(rf_model, 'random_forest_model_compressed.pkl', compress=3)
```

2. **Versioning:** To avoid issues with model updates, we included version numbers or timestamps in the filenames. Example:

```python
#Definition of version number
model_version = "v1.0"

#Create a file name using the version
filename = f'random_forest_model_{model_version}.pkl'

#Save model using version number
joblib.dump(rf_model, filename)
print(f"Model saved as {filename}")
```

3. **Environment Compatibility:** Ensuring that the versions of libraries like scikit-learn and NumPy remain consistent between training and deployment environments to avoid compatibility issues.

3. Model Serving

The serialized model is loaded and served using Flask, a lightweight web framework. This setup allows the model to accept input data and return detection. And this involved integrating the serialized model into a lightweight framework that supports quick response times and scalability.

Model Serving Process

1. **Loading the Serialized Model:** The model is loaded during the application's initialization phase to ensure it is readily available for predictions.
2. **Receiving User Input:** The server accepts HTTP POST requests containing the input data in JSON format.
3. **Making Predictions:** Upon receiving a request, the server processes the input data, applies necessary preprocessing, and uses the loaded model to generate predictions.
4. **Returning the Output:** The predictions are sent back to the user in JSON format, ensuring compatibility with various client applications.

[19]:
```python
from fastapi import FastAPI
from pydantic import BaseModel
# import joblib
# import numpy as np

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

# Initialize FastAPI app
app = FastAPI()

# Define input data format
class ModelInput(BaseModel):
    features: list

@app.post("/predict")
def predict(input_data: ModelInput):
    input_array = np.array(input_data.features).reshape(1, -1)
    prediction = model.predict(input_array).tolist()
    return {"prediction": prediction}
```

Cloud vs. On-Premises Deployment

When considering the deployment environment, we evaluated two primary options:

Cloud Deployment:

Advantages: Scalable, cost-effective for varying workloads, easy to maintain, and accessible globally.

Disadvantages: Requires reliable internet connectivity and ongoing subscription costs.

On-Premises Deployment:

Advantages: Full control over hardware and data security, no reliance on internet connectivity.

Disadvantages: High upfront costs, limited scalability, and requires dedicated IT management.

4. API Integration

The Flask API exposes a /predict endpoint that accepts JSON input and returns detection. To facilitate easy access to the machine learning model, we designed a REST API that interacts with

the model for real-time predictions. The API was structured to handle input data, process it, and return predictions in a user-friendly format.

API Details

1. **Endpoint:**
    - `/predict:` Handles prediction requests.
2. **Input Format:**
    - JSON object containing feature values.

```
#Input Format
{
  "Symptoms": ["Fever", "Cough", "Shortness of breath"],
  "Age": 35,
  "Sex": "Male",
  "Disease": "Respiratory Infection",
  "Treatment": "Antibiotics",
  "Nature": "Severe"
}

```

5. Web Application using Streamlit

To make the model accessible via a web interface, the Streamlit library is used to build an interactive web application where users can input data and receive predictions in real time.

Installation:

```
!pip install streamlit
```

Streamlit App Code:

```python
import streamlit as st
import joblib
import numpy as np

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

st.title("Disease Detection Based on Symptoms")
st.write("Enter symptoms and get a prediction.")

# Create input fields (adjust based on your model features)
input_features = []
for i in range(5):  # Change 5 to the number of features in your model
    input_features.append(st.number_input(f"Symptom {i+1}", value=0.0))

# Predict button
if st.button("Predict"):
    input_array = np.array(input_features).reshape(1, -1)
    prediction = model.predict(input_array)
    st.success(f"Predicted Disease: {prediction[0]}")

```

6. Security Considerations

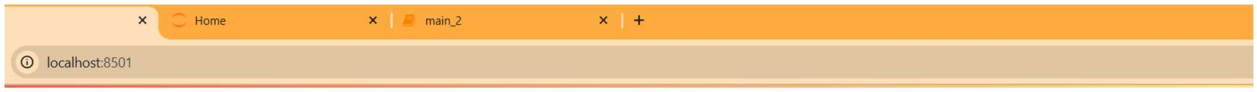
To secure the deployment:

- **Authentication**: Implement API key-based authentication to restrict unauthorized access.
- **Data Encryption**: Use HTTPS for secure communication.
- **Input Validation**: Validate incoming data to prevent injection attacks.

7. Monitoring and Logging

To track model performance:

- **Logging**: Requests and responses are logged using logging in Python.
- **Performance Metrics**: Accuracy, precision, and recall are periodically evaluated.
- **Alerting**: Set up alerts for unexpected behavior using monitoring tools like Prometheus.



Early Detection of Disease Based on Symptoms

Enter symptoms and get a prediction.

Symptom 1

0.00 − +

Symptom 2

0.00 − +

Symptom 3

0.00 − +

Symptom 4

0.00 − +

Symptom 5

0.00 − +

Predict